$2.50 USA

Australia	A $ 4.00	New Zealand	NZ $ 4.00
Singapore	S $ 8.00	Hong Kong	H $20.00
Malaysia	M $ 8.00		

'68' MICRO JOURNAL

VOLUME III ISSUE VII • Devoted to the 68XX User • July 1981

"Small Computers Doing Big Things"

SERVING THE 68XX USER WORLDWIDE

UniFLEX™

Multi-User

UniFLEX is the first full capability multi-user operating system available for microprocessors. Designed for the 6809 and 68000, it offers its users a very friendly computing environment. After a user 'logs-in' with his user name and password, any of the system programs may be run at will. One user may run the text editor while another runs BASIC and still another runs the C compiler. Each user operates in his own system environment, unaware of other user activity. The total number of users is only restricted by the resources and efficiency of the hardware in use.

Multi-Tasking

UniFLEX is a true multi-tasking operating system. Not only may several users run different programs, but one user may run several programs at a time. For example, a compilation of one file could be initiated while simultaneously making changes to another file using the text editor. New tasks are generated in the system by the 'fork' operation. Tasks may be run in the background or 'locked' in main memory to assist critical response times. Inter-task communication is also supported through the 'pipe' mechanism.

Support

The design of UniFLEX, with its hierarchical file system and device independent I/O, allows the creation of a variety of complex support programs. There is currently a wide variety of software available and under development. Included in this list is a Text Processing System for word processing functions, BASIC interpreter and precompiler for general programming and educational use, native C and Pascal compilers for more advanced programming, sort/merge for business applications, and a variety of debug packages. The standard system includes a text editor, assembler, and about forty utility programs. UniFLEX for 6809 is sold with a single CPU license and one years maintenance for $450.00. Additional yearly maintenance is available for $100.00. OEM licenses are also available.

FLEX™

UniFLEX is offered for the advanced microprocessor systems. FLEX, the industry standard for 6800 and 6809 systems, is offered for smaller, single user systems. A full line of FLEX support software and OEM licenses are also available.

technical systems consultants, inc.
Box 2570, West Lafayette, IN 47906
(317) 463-2502 Telex 276143

'68' MICRO JOURNAL

Portions of text prepared using the following.

SWTPC 6800-6809-DMAF2-CDS1-CT82-Sprint 3
Southwest Technical Products
219 W. Rhapsody
San Antonio, Texas 78216

EDITOR - WORD PROCESSOR
Technical Systems Consultants, Inc.
Box 2573, W. Lafayette, IN 47906
FLEX is TM of TSC

GIMIX Super Mainframe-Assorted memory boards
GIMIX Inc.
1337 West 37th Place
Chicago, Il 60609

CONTENTS

Send All Correspondence To:

'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixson, Tennessee 37343
— Phone —
Office: 615-870-1993

'68' Micro Journal is published 12 times a year by '68' Micro Journal, 6131 Airways Blvd., Chattanooga, TN 37421. Second Class postage paid at Chattanooga, TN. Postmaster: Send Form 3579 to '68' Micro Journal, PO Box 849, Hixson, TN 37343.

1-Year $18.50 2-Year $32.50 3-Year $48.50

—ITEMS SUBMITTED FOR PUBLICATION—

(Letters to the Editor for Publication) All 'letters to the Editor' should be substantiated by facts. Opinions should be indicated as such. All letters must be signed. We are interested in receiving letters that will benefit or alert our readers. Praise as well as gripes is always good subject matter. Your name may be withheld upon request. If you have had a good experience with a 6800 vendor please put it in a letter. If the experience was bad put that in a letter also. Remember, if you tell us who they are then it is only fair that your name 'not' be withheld. This means that all letters published, of a critical nature, cannot have a name withheld. We will attempt to publish 'verbatim' letters that are composed using 'good taste.' We reserve the right to define (for '68' Micro) what constitutes 'good taste.'

(Articles and items submitted for publication) Please, always include your full name, address, and telephone number. Date and number all sheets. TYPE them if you can, poorly handwritten copy is sometimes the difference between go, no-go. All items should be on 8X11 inch, white paper. Most all art work will be reproduced photographically, this includes all listings, diagrams and other non-text material. All typewritten copy should be done with a NEW RIBBON. All hand drawn art should be black on white paper. Please no hand written code items over 50 bytes. Neatly typed copy will be directly reproduced. Column width should be 3¼ inches.

(Advertising) Any Classified: Maximum 20 words. All single letters and/or numbers will be considered one (1) word. No Commercial or Business Type Classified advertising. Classified ads will be published in our standard format. Classified ads $7.50 one time run, paid in advance.

Commercial and/or Business advertisers please write or phone for current rate sheet and publication lag time.

Does timesharing on a small system make sense?

It does with OS-9™ Level One!

Now two (or more) acts can share your microcomputer stage. You will no longer have to walk away from your computer while it is busy running a long program. Because OS-9 is a multitasking operating system, you can be running a BASIC program while editing a PASCAL program, for example. This lets you make more efficient use of your time and your system, even if you only use one terminal. If your application requires multiple, independent terminals, one OS-9 system can do the work of several single-user systems.

The convenience of an advanced operating system

Sophistication does not require complexity. Many OS-9 users say that it is actually easier to use than the older 6800-type operating systems. Consider how easy it is to run multiple programs: to run a program you just type its name and hit 'return.' To run a program as a separate job, you type its name, an '&' character, then hit return. The program runs as usual, but OS-9 comes back immediately and is ready for your next command. Simple commands let you see each program's status, set its priority, or abort it.

The file management system has fast, byte-addressable random-and sequential-access files. The tree-structured multiple directory system lets you create separate disk directories for each user, project, or application. Command line I/O file redirection means you specify what device and/or files a program will use when you run it, not when you write it.

Efficiency and hardware versatility

No other operating system can run on such a broad range of hardware: the overall RAM requirement for Level One is 32K to 56K RAM. Memory utilization is superlative because OS-9 lets multiple tasks "share" the same reentrant program. For example, if two users run BASIC09, only one "copy" is actually loaded into memory. The Level Two version of OS-9 can utilize up to a megabyte of memory on systems having memory management hardware (both versions come with complete timesharing support).

OS-9's device independent I/O system can handle almost any number and combination of I/O devices: five or eight inch diskettes, winchester disks, disk cartridges, serial and parallel ports, memory-mapped video displays, and more.

Microware® offers a large selection of "stock" device interface software modules, or you can create your own: all the information you need is in the manuals.

Excellent support and documentation

Each OS-9 package comes with a User's Manual and a System Programmer's Manual that cover every aspect of OS-9. If you have special requirements, you can even purchase the Source Code for most of OS-9 and related software. At Microware® we take pride in offering the best customer support in the business. Technical advice and assistance by phone, mail or telex is available during all business hours.

Superb software tools

In addition to BASIC09, Microware® offers: an Interactive Assembler, Macro Text Editor, Stylograph Word Processor, Interactive Debugger, and coming soon, COBOL, PASCAL and C language compilers.

Microware® software is available for most popular 6809 computer systems. Source listings and yearly maintenance update service are sold separately for most programs.

Please call or write for out free catalog. We accept phone orders and MasterCard and VISA orders.

MICROWARE®

Microware Systems Corporation
5835 Grand, Des Moines, Ia 50312
(515) 279-8844 TWX 910-520-2535

X-FORTH FLEX COMPATIBLE FORTH

By Charles (Chuck) Eaker, Ph.D

"This is obviously the most comprehensive manual that's ever been produced on FORTH. It's vastly more complete than anything else! The way he talks about things is not only good reading, but he makes it easy to pick up on the first try."

Said Ron Anderson, '68 MICRO JOURNAL's contributing editor, talking about the X-FORTH manual.

X-FORTH is the best FORTH there is for 6809/6800 computers running FLEX! There's no better way to put it. X-FORTH beats the competition hands down and here's why.

FLEX COMPATIBLE
She can read and write FLEX random and sequential files. She can even read and write the sequential files RANDOMLY! Uses FLEX I/O for terminal and printer. Honors TTYSET.

TWO EDITORS
She has a TTY editor modeled after the FORTH INC. editor rather than the FIG version.
She has a FULL SCREEN EDITOR for terminals that support cursor addressing.

STANDARD ASSEMBLER
She uses standard MOTOROLA menomics thus:
'LDA [44]' becomes '[[44] LDA' in X-FORTH
6809 assembler also supports 6800 menomics!

ERROR CONTROL
You the programmer have complete control over disk related errors while other errors provide long error messages. X-FORTH has a protected dictionary and is very hard to accidentally crash!

FANTASTIC DOCUMENTATION (We're very proud of this)
4 Part Ring Bound Manual (over 400 pages!)

1... **GOING FORTH**
Tutorial on the use of FORTH that makes it a snap to learn!

2... **GOING X-FORTH**
X-FORTH extensions and FLEX interface.
Shows you what the 'X' in X-FORTH is all about!

3... **USER'S MANUAL, DOCUMENTATION AND GLOSSARY**
This is the main part of X-FORTH with sections on: math operations, stacks, input words, output words, strings, disk I/O, edit, editor, assembler, utilities, etc. ALL grouped by function for ease of use! Glossary listing included with each section.

4... **GLOSSARY AND SOURCE LISTING**
Glossary sorted alphabetically in addition to that in the users manual. Source of all but the core of X-FORTH (also included on disk!)

String primitives. Complete DATA FILE VOCABULARY. Triple precision math package. Portability between 6800 and 6809 versions of X-FORTH. Compatible with FIG FORTH. AND, THERE'S EVEN MORE WE DON'T HAVE ROOM TO LIST!

Supplied on one 8" Disk or 2, 5" disk(s) with a 400 page manual in a hard cover binder. Disk(s) have the source of everything but the core of X-FORTH, which will be available later at extra cost. You get it all!!!

We wanted to provide a better FORTH with more extras at less cost and we did it!!! Here it is!

All for only

Manual available separately for $49.95, with credit for later purchase. **$149.95**

WHAT ARE YOU WAITING FOR ...

NEW

BASIC PROGRAMMERS TOOLKIT

By Dick Bartholomew

$49.95 object only
$69.95 with source on DISK!

1...	DECOMPIL	Turns .BAC into .BAS files.
2...	XREF	Cross reference listing of BASIC programs.
3...	EDIT	Edit a BASIC program that's in memory while in BASIC!!!!!

NOTE: For TSC XBASIC only.

Programs are written in 6809 assembly language.

NEW

EXTENDED USE UTILITIES PACKAGE

By Dick Bartholomew

$49.95 object only
$69.95 with source on DISK!

1. *CRTSET like TTYSET for CRT's*
2. *USERINFO adds more than just diskname and number.*
3. *SCAN List forward and backward thru a file.*
4. *BROWSE Like SCAN but in memory*
5. *LOAD offset loader*
6. *DISKDUMP to any port.*
7. *INIT memory set command*
8. *SAVETEXT to disk*
9. *READTEXT from disk*
10. *REDIRECT change control port*
11. *REPLACE prompt memory change*
12. *MIRROR BACKUP fast sector copy*

Programs are written in 6809 assembly language.

NEW

PASSWORD PROTECTION PACKAGE

By Dick Bartholomew

$69.95 object only
$89.95 with source on DISK!

Six programs that allow you to protect your system by preventing a bootup without the proper password.

— Warning —

PASSWORD can be overcome if the user has access to the reset button or another disk without password protection that will boot FLEX.

ALSO AVAILABLE

THE BILL PAYER a home payables system By Jim Schreier $89.95
options P/O system $49.95
NEW Income/Expense ledger $49.95

See previous ads for other programs.

COMING SOON
PRO-EDIT a PROfessional editor for the PROgrammer.

U.S.A. add $2.50 for Standard UPS Shipping & Handling
Foreign orders add 10% Surface, 20% Airmail

DEALER INQUIRY ENCOURAGED. Contact Frank Hogg for more information.
ATTENTION PROGRAMMERS!
We are looking for quality software to market ... Contact Frank Hogg.

FRANK HOGG LABORATORY, INC. 130 MIDTOWN PLAZA • SYRACUSE, N.Y. 13210
(315) 474-7856

Dataman

DATABASE MANAGEMENT

MEANS BUSINESS

DATAMAN is made up of 15 menu-driven programs which make up a powerful sequential DBM system under FLEX. You can think of DATAMAN as a computerized card index. Use DATAMAN at home to keep track of your household possesions, record and tape libraries, mailing lists for cards and letters, etc. For your business, use DATAMAN for product inventory, newsletter subscriptions, marketing systems, check balance reconciliation and much more. The uses are limited only by your imagination. DATAMAN turns your 6800/6809 computer into a powerful database management system which can save you a great deal of time and money.

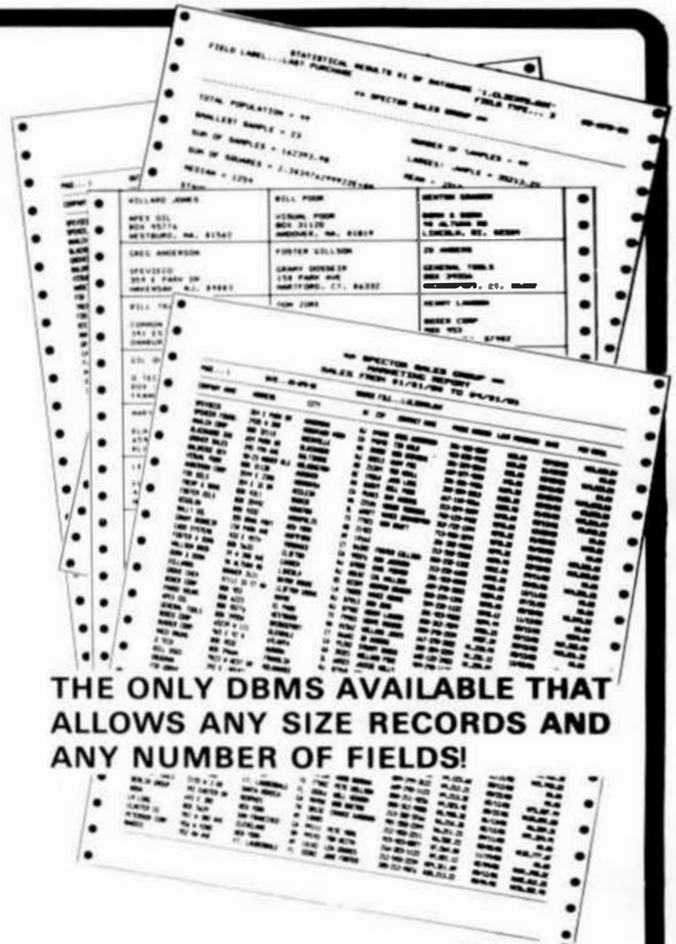
You can create databases and do maintenance. Report writing is easy with both vertical and horizontal formats. Both 80 and 132 column reports are supported in the horizontal format. Formats are saved with run time options available. Label printing is made easy with up to 3 easily changed label drivers on line, and as with other DATAMAN output programs, the output can be spooled to disk, for later printing. A statistics package gives up to 24 statistical values. You can transfer records from one database to another, blow away records, even merge two unlike databases together on a key. Sorting on up to 20 keys is done with a sort editor which uses the TSC Sort/Merge package. You can even build "PR" files for use with the TSC Text Processor for form letters, invoices, reports, etc. The list of features goes on and on!

You can create a database with any number of fields. Most database systems require you to limit the total data in one record to 254 characters or one sector. Not so with DATAMAN! DATAMAN allows your records to be of any length. We made DATAMAN to work for you ..... not against you!

DATAMAN was designed for easy modification and customizing with a "SHELL" program. A "SYSTEM GEN" file allows almost any terminal or printer to be used with DATAMAN. Comes with every line of source code, program and variable descriptions, and a 160+ page ring bound manual that is second to none!

$149.95

*FLEX is a trademark of Technical Systems Consultants, Inc.



THE ONLY DBMS AVAILABLE THAT ALLOWS ANY SIZE RECORDS AND ANY NUMBER OF FIELDS!

Datarand NEW

RANDOM ACCESS

If you work with databases containing more than 20 records and you need super fast look-up and editing, DATARAND is for you. DATARAND converts DATAMAN files to random files with up to 9 keys. No need to worry about sector blocking or record length. DATARAND takes care of everything! You can look up a record by key value, key wildcard, or record number. DATARAND uses hashed key files which means no slow ISAM searches. Editing is quick and easy using the same screen layouts as DATAMAN.

The same "Record Code" concept is used to make operation simple. In fact, except for the speed, it's hard to tell that you're not in DATAMAN. You can add records to DATARAND just like in the "Append" section of DATAMAN file maintenance. When you need to use DATAMAN, you can copy the desired records to DATAMAN. You can also transfer records from DATARAND which takes care of file reorganization. DATARAND comes with an instruction manual and every line of source code.

$49.95

U.S.A. add $2.50 for Standard UPS Shipping & Handling
Foreign orders add 10% Surface, 20% Airmail
DEALER INQUIRY ENCOURAGED. Contact Frank Hogg for more information.
ATTENTION PROGRAMMERS!
We are looking for quality software to market ... Contact Frank Hogg.

Datacalc Coming Soon!

NDEFGHIJKLMNOPQRSTUVWXYZ,/; [@^]-1()=}~&

Epson Factory Authorized Repair Center

SOUTH EAST MEDIA

P.O. Box 794 Chattanooga TN 37443
1-615-877-2241

10 CPI Standard

ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
abcdefghijklmnopq
1234567890123456
EPSON MAKES MORE
Epson makes more

10 CPI Emphasized

ABCDEFGHIJKLMNOP
abcdefghijklmnopq
12345678901234567
EPSON MAKES MORE
Epson makes more

5 CPI Double Emphasized

ABCDEFGH
abcdefgh
12345678
EPSON MA
Epson ma

For most people, spending money for a printer with capabilities they'll never use just doesn't make sense. That's why we designed the MX-70.

The MX-70 is the lowest-priced 80-column dot matrix printer on the market today. But it's still one fine little printer. In fact, the MX-70 has features that, until recently, you couldn't buy at any price.

One of the family.
The MX-70 is one of the Epson MX Series of printers. It shares technology based on years of intensive research and development. Its patented, disposable print head prints between 50 and 100 million characters. When it wears out, just throw it away. It's user replaceable at about 1/3 the cost of standard print heads. Our precision plastic injection molded case eliminates most costly hand assembly, which keeps the price low and the quality high.

Everything you need.
The MX-70 prints unidirectionally at 80 CPS with a user-defined choice of 40 (double width characters) or 80-column printing. It gives you top of forms specification, programmable line feed and form lengths, an easy-to-read 5x7 matrix, skip feed, and an adjustable tractor feed. And because it's a true impact printer, it doesn't need costly aluminized or thermal paper. Plain paper is just fine.

And we give you graphics at no extra charge.

GRAFTRAX II is one of the finest bit-plot graphics capabilities on the market today. And on the MX-70 it's free. With unparalleled dot placement accuracy, the bit image graphics are free of the jitter, wander and walk that plague most other printers. So you get blacker blacks, cleaner grays and definition you just won't find anywhere else. That's not a bad bonus on a basic little printer, eh?

No-frills printing at a no-frills price.
If you want a simple, no-frills printer; if you don't want to buy features you'll never use; if you want a reliable printer backed by the world's largest manufacturer of print mechanisms, we're afraid there's only one logical choice.

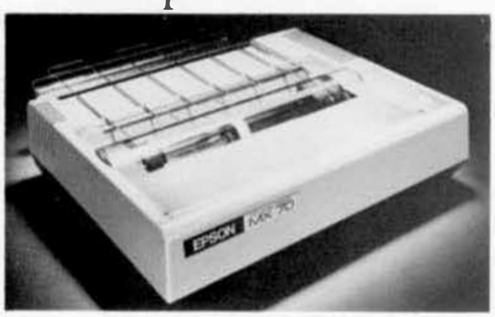
The Epson MX-70

IN THE WORLD.
in the world.

IN THE WORLD.
in the world.

CHANISMS
chanisms

The Epson MX-70



$375.00

PRINTING CHARACTERISTICS · PRINTING SIZES · FORMS HANDLING · MEDIA HANDLING · INTERFACES · SWITCHES LIGHTS DETECTORS · RELIABILITY · INKED RIBBON · ENVIRONMENTAL CONDITIONS · POWER REQUIREMENT · SELF TEST MODE · PHYSICAL CHARACTERISTICS

The Epson MX-80

$495.00

INTERFACES · SWITCHES LIGHTS DETECTORS · PRINTING CHARACTERISTICS · RELIABILITY · INKED RIBBON · ENVIRONMENTAL CONDITIONS · POWER REQUIREMENT · SELF TEST MODE · FORMS HANDLING · MEDIA HANDLING · PHYSICAL CHARACTERISTICS

Specifications subject to change without notice.

This is a South East Media Special to 68 Micro Journal Readers.

Add $7.50 Shipping and Handling, Allow stock to 3 weeks.

abcdefghijklmnopqrstuvwxyz!!"$12345

NOTICE

It has lately come to our attention that one of our past advertisers is experiencing some difficulty in honoring orders submitted by our readers and others. It is the policy of 68 Micro Journal to alert our readers to any possible problems, of a serious nature, that may arise with any of our advertisers. In view of the above the following is published for your attention:

Numerous complaints have been received, by 68 Micro Journal, in the past month or so, concerning the following past advertiser:

Southwest Microsystems
6803 Kings Point Lane
Austin, Texas
78723

Should this situation be resolved, to the satisfaction of the complaining readers, and we are properly notified, we will inform our readers, by a like notice in 68 Micro Journal. Until such time it is recommended that any reader dealing with the above named company understand the contents of the above.

Don Williams Sr., Publisher

- - - - -

Flex User Notes

BY: RONALD W. ANDERSON
3540 STRUBRIDGE COURT
ANN ARBOR, MI 48105

ON TESTING PROGRAMS

I've been evaluating a lot of software recently, and a section of a book I have keeps coming to mind. I have received permission from Winthrop Publishers to quote it for you. It deals with the various levels of testing of a program. Some of us don't know how to test our programs. Some don't care to test them. If, however, you are going to sell a program and you don't want to invite disaster, you had better spend some time testing your efforts. The following notice is required by Winthrop:

From A PRIMER ON PASCAL, first edition by Richard Conway, David Gries, and E. Carl Zimmerman. Copyright (c) 1976. WINTHROP COMPUTER SYSTEMS SERIES. Reprinted by permission of Winthrop Publishers, Inc., Cambridge, Massachusetts.

THE MEANING OF CORRECTNESS

'"Program correctness" is not easily defined. The programmer and user of a program may discover they use quite different meanings of the word "correctness", and hence have quite different expectations of program performance. Consider the following possible interpretations of correctness — listed in order of increasing difficulty of achievement:

1. The program contains no syntax errors that can be detected during translation by the language processor.

2. The program contains no errors, either of syntax invalid operation, that can be automaticaly detect during translation or execution of the program.

3. There exists some set of test data for which t program will yield the correct answer.

4. For a typical (reasonable or random) set of test da the program will yield the correct answer.

5. For deliberately difficult sets of test data t program will yield the correct answers.

6. For all possible sets of data which are valid w respect to the problem specification, the progr yields the correct answers.

7. For all possible sets of valid test data, and for likely conditions of erroneous input, the program give correct (or at least reasonable) answer.

8. For all possible input, the program gives correct reasonable answers.

In the early stages of your programmi experience you will feel harassed by error messag during translation, and feel a sense of achieveme when you have attained level 1 correctness. Howev the absence of error messages is only a necessary a not a sufficient condition for reasonable correctne You will eventually regard the detection of such erro as a helpful service, which unfortunately detects o the easiest of errors.

Some students never mature beyond level 3 f an interpretation of correctness. We are regula involved in arguments challenging the grade assigned a problem on grounds that it "worked" on the studen own data, hence must be correct. In effect, the stude is arguing that level 3 is adequate. Considering high levels (say 4, 5 or 6) it is clear that satisfacto performance on any single set of test data is n sufficient grounds for an assertion of correctness, b failure on a single test is sufficient to demonstra that the program is not correct. NO MATTER HOW MA TESTS THE PROGRAM MAY HAVE PASSED SUCCESSFUL JUST ONE TEST ON WHICH IT FAILS IS ENOUGH TO SH THAT IT IS NOT CORRECT. This is not inherently democratic process, and a program tha works "most the time" is a dangerous tool.

From the "customer's" point of view reasonable definition of correctness is certainly n less than level 6, while the customer will maintain th certain implied requirements do not have to explicitly stated. In effect, this corresponds to t "implied warranty of merchantability" that accompani a manufactured product. A consumer is entitled assume that a product is "suitable for the prupose f which it is intended". A car buyer, for example, c rightfully assume that all the wheels will remain fir attached to the car, without having to obtain a writt guarantee from the dealer. In the same way, much assumed about a computer program, without its havi been explicitly detailed in the problem requirement The user of a program is entitled to consider incorrect if it fails to satisfy implicit as well explicit requirements.

Unfortunately, this often leads to heat discussions between programmer and user, the obje being to assign blame for a program belatedly found be incorrect. The programmer takes the position th there is no such thing as implicit requirements; t user maintains that, in retrospect, anything neglected to specify is covered by implic commonsense requirements. Both parties should reall that implicit requirements are an inherent part of m problem descriptions, and that it is a MUTU responsibility to explore this subject to ensu

mutual understanding of context of use, nature of errors, appropriate reactions and communications.

The PRIMARY RESPONSIBILITY RESTS WITH THE PROGRAMMER. A program is INCORRECT IF IT DOES NOT SERVE THE USER'S PURPOSES. This may occur because the programmer failed to elicit an adequate description, because he failed to recognize implicit requirements, or because he made mistakes in designing or translating the algorithm into a programming language. Most programmers admit responsibility for only the last two sources of error, but the distinction between different types of failure is not interesting to a user with an unsolved problem.

In summary, the situation is the following. The user would like to have level 8 correctness — but this is usually impossible, and he might as well get used to that fact. Level 7 is a reasonable compromise, which is obviously going to lead to arguments since it leaves critical questions open to varying interpretations. The programmer's dilemna is that level 5 is the highest that can be achieved by purely empirical means — by running the program on test cases — so he must thoughtfully design test cases while writing the program that will permit a plausible assertion that level 6 has been achieved. To achieve level 7 the programmer must know enough about the intended use of the program to estimate what errors are likely to be encountered, and what response is appropriate."

End of quote. Quite a challenge to all of us who call ourselves programmers in any sense of the word. I recently received some Business software for review. I discovered that by answering the prompt "FILENAME" with the name of an existing file, without warning the existing file was deleted and replaced with the new file. In my opinion, that indicates failure of the program at level 4. I think Business users have a right to expect more than this. There is some very good software available, but a small business owner/user who with some misgivings has bought a Microcomputer system and has purchased software that has been debugged to level 4, will soon give up and tell all his friends that small computers are no good for small business applications.

Along this line I can report some very encouraging news. In addition to the software mentioned above, I've purchased software from another supplier, and had a couple of things sent me for "preview" from a third supplier. I can truthfully say that I have not been easy on those suppliers in terms of my reaction to bugs that I have found. I must say that their response has in all three cases been very favorable. I'm sure that I've irritated some of the suppliers with my negative reactions, but all have indicated that my complaints were for the most part valid, and ALL HAVE MADE CORRECTIONS AND SUPPLIED ME WITH UPDATED SOFTWARE. Further, I must say that the responses have been very timely. One of the authors of a particular Program called to tell me about the fixes, mention some improvements that are coming, and to say that surely others have found some of these problems, but that no one had reported them. I was thanked for responding and helping to get the software debugged.

Software is getting more sophisticated these days. The authors/suppliers point out that in general they have little way to imagine the uses to which their products will eventually be put. They have, in my experience been unanimous in wanting to have the best product possible. They just need to know when a bug is found, or a feature is needed.

I do, however think that more thorough testing of products before they are offered to the public, would result in more favorable initial reaction to the products. This therefore is a plea for more thought in testing of your products before their release. I earn a living working for a company with a product. I know that companies starve or go broke if they don't ship their wares regularly. I also know that companies go broke when warranty expenses exceed profits, or their reputation is wrecked by customer dissatisfaction. It is inevitable that customers will be the final testers of software, and none of us minds an occasional obscure problem with new software, if the supplier is responsive. What constitutes good software, is a subjective sort of thing with regard to features and operator interface. Whether the software is debugged or not, is a rather more objective thing. Consumers don't want to be the ones to find gross problems in new software.

All of us who program for a living or a supplement to our living need to take this to heart and try to produce software that reaches level 6 plus. If I've intrigued any of you with the contents of A PRIMER ON PASCAL, a new edition is now available. Write Winthrop Publishers, Inc. 17 Dunster St., Cambridge, MA 02138 for further information on price and availability. The whole book is written in this manner. It is much more than just another book on using another programming language. It is in fact, a companion book to a volume by the same authors called "A Primer on Structured Programming — Using PL/1, PL/C, and PL/CT". It contains a great deal of information on programming by "stepwise refinement", an approach whose principles are language independent.

NEW THINGS

I recently received a notice from TSC that their Sort Merge package is now available for UNIFLEX.

CALIFORNIA CORRECTIONAL INDUSTRIES

Because of this column, I run across interesting people and stories. This one is intimately related to computing, so here goes. In February, I received a letter from Dennis Herk who is a "resident" of California State Prison at San Luis Obispo. Dennis sent a most helpful letter regarding using active terminations to cure SS-50 Bus intermittents. I answered his letter, and the next one from him brought a reprint of an article in the Prison Newspaper, and what I would call a product brochure. I've asked permission to write this up, and received it from Dennis and his Supervisor.

CCI employes about 400 of the 2400 inmates at the facility. There are various plants such as a Knitting Mill, Laundry, Shoe Factory, Textile Plant, and Printing Plant. Dennis is employed in Maintenance, and has been involved in the design and construction of some controllers for the various machines in CCI. The brochure describes a Dryer Control that measures moisture content of the contents of the Laundry Dryer, and shuts it off at the correct point to conserve energy and maximize "production". It also describes a Washer Polling Control that conserves energy and reduces water pressure fluctuation by allowing only one washer to fill at a time. This arrangement also reduces and evens out the load on the water heater. The brochure goes on to say that the control has shown an average 6% saving in energy required for water heating. It concludes with "These control units plus other electronic/electromechanical devices designed by California Correctional Industries, are offered for sale to public agencies. Our design team is open to the challenge of developing similar automatic controls and other devices to meet your specific requirements."

Dennis has a 6800 system running and is "teaching the managers basic operation during lunch." He finds it difficult to convince people there that Microcomputers can achieve the throughput of a "mini" (Just as the rest of us do). I'd like to close this little section with a quote from Dennis and a comment. Dennis

says "I'm not...ashamed of making a mistake. Now if I let greed guide me again into one of these cesspools, then shame would be apropo." Dennis, with an attitude like that, you won't have to worry about seeing the inside of one of those places again!

MORE ON PRIMES PROGRAM OPTIMIZATION

Last column I wrote about the virtues of thinking a little about optimizing the Algorithm for improving speed of execution of a program. My example was the Prime program, and I reported a time of 1 minute and 12 seconds for the primes to 10000 with Omegasoft Pascal. The other night I was looking at the program and decided that I had lots of multiples to compare the squares of my test numbers to the number being tested, in order to terminate my testing at the optimum point. Since I have an array of the first primes found, I decided to generate a second array of their squares. That way, the multiplication would only have to be done once for each of the primes found that are less than the square root of the maximum number. I wondered if the additional array access would take more time than the multiplications. I found out something very interesting. The added array has a dimension of 50 which means that it only takes 100 bytes of RAM at runtime, so that is not much of a disadvantage. The run time for Primes to 1000 was virtually the same as the original program. However, as the numbers being tested get larger and larger, more multiplications are eliminated, and the array access time begins to be significantly less than the time spent previously multiplying. The result is that for primes to 10000, the execution time is reduced to 1 minute and 6 seconds, about 8.3% improvement over the old program. I had tried similar optimizations before, but hadn't tested them on the larger limit. For your amusement, I have included the listing for the program in Pascal here, (The "most optimum" version).

Perhaps someone out there will find another factor of 2 or 3 in time reduction by seeing the obvious that I am missing. While I am at it, I would like to mention another example of optimization. The application is rather complex, but essentially, it is a program that takes all possible combinations of several variables and finds the "best set" by some criteria that are dictated by the problem. In this case, the problem was a mechanical part with 8 possible locations for balance weights, and each location could have no weight or one of 6 fixed sizes of balancing weights. The idea is to find the smallest set of weight values and positions that will adequately correct any unbalance amount and position within some maximum limit. I had written a program some time ago for a different though similar application, and managed to get it to run in about 2 hours. A more complicated problem came along, and following my method for it resulted in a program that ran about 50 hours. A friend took the problem home for a week-end of sitting in front of the computer, and came up with another approach. His program runs in about 20 minutes and produces a very good set of solutions to the problem. Sometimes a fresh approach can result in drastic savings in computer time.

There was a net gain here, since the program didn't take my friend Mike Myers anywhere near 50 hours to write and debug. Further, the approach is a valid one for future problems of a similar nature. Mike found several interesting approaches to reducing run time. First he scaled all the data so he could use integer variables in his BASIC. Secondly, he started with the worst case in terms of using excessive numbers and sizes of weights to make corrections. If any solution was found later in the program that "covered" the same unbalance area as an older one, it had to use less weights and therefore be a better solution. Mike therefore cleverly eliminated a lot of comparisons to see if a new value was "better" than an older one. Mike also used several large arrays to hold all the values necessary for the calculation, therefore eliminating the necessity for repeatedly calculating, for example, the Sine of 57 degrees, etc. Overall, the solution was a very good one. This is particularly impressive to me since Mike is not an engineer. His job is in Sales. He has had a computer of his own for a couple of years now, and started like most of us at ground zero at that time. Anyone out there have some further concrete examples of such things?

FLASH: April '68' just arrived. In it, Dave Shirk of TSC made some comments about the Moreira article in Feb. '68'. Included is an algorithm for finding the primes by Wirth. Dave reports the time to write the numbers to their terminal at about 10 seconds. They must be running 9600 baud. I checked my write time and arrived at 5 seconds at 19,200 baud. Since a copy of Lucidata's Pascal release 3.9 arrived today also, I dumped in the Wirth algorithm program and ran it in Lucidata and Omegasoft Pascals WITHOUT ANY CHANGES OF ANY KIND!! Adjusting Dave Shirk's time to my 19K baud (so we can compare apples and apples), apparently TSC Pascal will run the program in under 11 seconds. Omegasoft did it in 34 seconds (About twice as fast as my most recent effort reported above), and Lucidata took 125 seconds. Lucidata, incidentally, ran my latest version in 156 seconds. Next month, I will report on the new features of Lucidata Pascal, and perhaps try to explain the Wirth algorithm for the Primes, and why it is again more efficient. Dave's listing does contain one non-fatal error, the correction of which makes the program even more impressive. Down near the end of the listing there is a statement 'IF I <= NL THEN P[I]:=X;' Therefore, no assignments are made to the P array beyond the subscript equal to NL. The VAR declaration should therefore be P,V : ARRAY[1..NL] OF INTEGER;. This reduces the arrays to dimension 35 rather than 1229. This doesn't change the execution time, but the RAM required for the data stack is much less. Just incidentally, some authorities include 1 as a prime number and others do not. You will find that my program listed here includes 1 and finds 1230 primes between 1 and 10000.

As soon as I can check out TSC Pascal, I will be reporting on it too. Apparently it is very fast. I'd like to pose a question to Dave Shirk regarding his comments on maximum use of the 6809 features. Dave, I can see that having BASIC in position independent code might be a big advantage with Uniflex in a multi-user situation in which each user is allotted his "chunk" of memory, but what advantage is it to the large number of users that have FLEX9 single user systems, and are not even aware of how (nor have they need) to load it anywhere else in memory? I fully agree that speed isn't the only important parameter in evaluating software, and TSC is to be commended for planning ahead and making BASIC compatible with the multi-user system by taking advantage of the position independent code possible with the '09', but it is a feature that many of the users will not be taking advantage of.

```
PROGRAM PRIME (INPUT,OUTPUT) ; { FIND PRIMES TO MAXNUMBER }

VAR
   NUMBER    : INTEGER ; { CANDIDATE FOR PRIMALITY TEST }
   MAXNUMBER : INTEGER ; { UPPER LIMIT OF PRIMES TO BE FOUND ;
   ITEM      : INTEGER ; { COUNTER FOR PRIMES FOUND }
   K,L       : INTEGER ; [ LOOP INDICES }
   COLUMN    : INTEGER ; { FOR OUTPUT FORMATTING }
   PRIM      : ARRAY [1..50] OF INTEGER ; { SAVE PRIMES TO SQRT(MAXNUM) }
   PRIMSQUR : ARRAY [1..50] OF INTEGER ; { PRIMES SQUARED }
   SWITCH    : BOOLEAN; { SWITCH FOR SAVING OF PRIMES }
   ISPRIME   : BOOLEAN; { SIGNAL THAT PRIME WAS FOUND }

BEGIN
   PRIM [1]:=1; PRIM [2]:=2; PRIM [3]:=3;
   PRIMSQUR [1] := 1; PRIMSQUR [2] := 4; PRIMSQUR[3] := 9;
```

```
   WRITELN ('INPUT UPPER LIMIT OF PRIMES');
   READ (MAXNUMBER) ;

   WRITE (PRIM [1] : 7,PRIM [2] : 7,PRIM [3] : 7) ;

   { INITIALIZE VARIABLES AND POINTERS }

   ITEM:=3; COLUMN := 3; NUMBER:=5; L:=4; SWITCH:=TRUE;

   WHILE NUMBER <= MAXNUMBER DO
   BEGIN
     ISPRIME:= TRUE; K:=3; { ASSUME NUMBER IS PRIME UNTIL PROVEN NOT }

     WHILE (PRIMSQUR [K] <= NUMBER) AND ISPRIME DO
     BEGIN
       IF NUMBER MOD PRIM [K] = 0 THEN ISPRIME:= FALSE;
       K:=K+1;
    END; {WHILE PRIMSQUR}

     IF ISPRIME THEN
     BEGIN
       WRITE (NUMBER : 7);
       ITEM:= ITEM+1;
       COLUMN := COLUMN + 1;
       IF COLUMN = 10 THEN
       BEGIN
         WRITELN;
         COLUMN := 0
       END;
       IF SWITCH THEN
       BEGIN
         PRIM [L]:= NUMBER;
         PRIMSQUR [L] := NUMBER * NUMBER ;
         IF PRIMSQUR [L] > MAXNUMBER THEN SWITCH:= FALSE;
         L:= L+1;
       END; {IF SWITCH}
     END; {IF ISPRIME}
     NUMBER:= NUMBER+2;
  END; {WHILE NUMBER}
  WRITELN ;
  WRITELN('THERE ARE ',ITEM,' PRIMES BETWEEN 1 AND ',MAXNUMBER);
END. (PRIME)
```

febe

While attending the 'Philly '80' show last summer, I was impressed with a Standard S50 Bus main frame kit being shown. Later we received, for review, a kit including the Cherry Pro keyboard. This is a report of the feber group, Inc. - ECB-50 kit.

The kit came with the following units. Case - A two piece injection molded styrene case with provisions for a cooling fan. The case is furnished unpainted for custom finishing (we painted ours white for about $2.50 in spray paint). Keyboard and portal cutouts are provided. Rubber feet and a smoked plexiglass front panel is standard.

Power Supply - 15 amp @ 9 volts and 2.5 amp @ +/- 17 volt. We find this was adequate for fully populating the motherboard and also a pair of 5 1/4 inch disks. All voltages appear at a barrier strip.

Motherboard - Tinned double sided PC card with plated through holes. Sockets provided for all IC's. All address and data lines are terminated. User selectable port size (4,8,16, etc). Base address of ports addressable $8000 to febe+ hex (you name it). Seven 50 pin slots and eight 30 pin I/O slots.

Keyboard - Cherry Pro with 67 keys. 128 ASCII codes, 5 volt only supply. Five user definable keys and four mode operation. Repeat option available.

When we received ours the shipping policy was that febe shipped the manual about a week before the mainframe kit. This allows time for leisure study and planning for the construction period. Over the years I have talked to many, many builders who stated that they wished that they had read the 'book' more and not been in such a 'darned fired hurry'! In our case we know it helped as the builder had never attempted to build ANY kit, and to top it off, he knew nothing about computers (then).

We wanted to see if it were actually documented sufficiently to allow a novice an even break in getting things right. It did and he did, fired up first time and no problems. The manual is very complete and there was only one major (or minor depending on skill level) problem in the documentation. An readily apparent difference between the power supply component placement drawing and the parts layout diagrams. A quick reference to the main diagrams resolved this difference. Otherwise, everything went together, and WORKED as indicated in the manuals. A revised (and accurate) drawing was received 2 days after the mistake was discovered. It is fair to note that the serial number of our kit was number 15. We have seen some kits that have been produced for years (and the serial number up in the thousands), yet they still have some serious mistakes in the manuals that can do a real thumping on you if you are not watching, and completely ruin your day (and project).

DOCUMENTATION

The manual consists of not only written instructions, but also full 'blue print' type technical drawings (circuit),logic and layout. Also included was an 8X11 slick photo of the system from two angles; parts placement and a look at how it is supposed to look when completed. The step-by-step instructions were in the proper order and no parts were missing.

An especially helpful section was the one headed 'SPECIAL NOTES'. This portion of the manual listed not only things to beware of in the construction and testing of the completed computer system, but also has some important tips on what to look for when populating the system with logic boards. Tips include: Port size selection, port address decoding, power supply loading, interfacing various cards and boards, 6809 slow peripherals and mention of an EPROM BASIC for this system and others. Also if 'all else fails' a telephone number is provided to allow the stumped builder immediate advice and technical help from the source. This is one of only a few who encourage telephone calls for assistance. Complete backup repair service is available if needed from febe.

The entire system is guaranteed for a period of 90 days from date of receipt.

THE FEBE GROUP

febe has organized a users group of technicians (factory and user) alike, engineers and others to assist kit builders and to pass along hints and kinks, to make life a little easier. This group meets regularly and promises to extend the help that some will need.

CONCLUSION

The quality of the materials and components furnished is commercial grade. The cabinet is both sturdy and attractive. The power supply is adequate and has more reserve than many more expensive systems. The documentation is complete and accurate. And we can truthfully say 'that a rank novice can put it together'.

Now that we have it running, we have some special plans for this system. It will be our CBB (community bulletin board) system. It will be connected to the land line by a THOMAS INSTRUMENTATION MODEM BOARD. A review on this at a later date, however, mention is made here that we have received many good remarks concerning this modem. The CPU will be a 68800, 2 mhz, running FLEX 2.0. We have all the hardware on hand. Currently we are attempting to get the necessary software to complete the hook up. This system, when fuctional, will allow 68 Micro Journal readers an access and message (program) swapping vehicle. We will keep you informed as we progress on this project.

Additional information can be secured from:

febe group, inc.
51 Hamilton Avenue
York, PA 17404
Phone 717 854-0481

EZDATA

John P. Tucker
POB 2898
Laredo, TX 78041

The user of the SS-50 bus is faced with probably the fewest in number of hardware problems of all the micro bus systems. He is also blessed with a wide variety of peripherals for his system. But he suffers from a dearth of good, usable business software.

Gary Magnusen of Lafayette, Indiana, has released a series of programs for data manipulation that close a big gap in the software line. Called EZDATA MANAGEMENT SYSTEM, the programs are aptly named. Making use of many of the capabilities of today's intelligent terminals, he has devised ways of creating files, sorting and manipulating them according to a wide variety of keys, and outputting them in formatted manner to either screen or printer.

Excellently conceived documentation accompanies the disk(s) to assist a programmer in configuring the programs to his own system. It is not necessary to tamper with the program proper as all configuration is done through a menu system. The programs arrived configured for Gary's Micro-Term Mime-1 terminal. Despite my unfamiliarity with my own brand new SWTPCo 8212, it took me about ten minutes to make the necessary changes to have the programs up and running.

And run, they do! Running under Flex 2.0, every step I tried worked without flaw. EZCTL creates and updates an EZDATA CONTROL file that acts as a job control system. This allows just a few key strokes to set up the most complex sort program, for instance, and execute it without operator intervention. EZDATA creates an EZDATA file, the basic file used throughout for input. EZTEXT creates a text file from an EZDATA file with essentially no effort on the users part. EZVIEW makes it possible to read what has been created. The files are not put on disk in a manner that the Flex LIST utility can be used to read them but the EZVIEW system puts them on the screen just as you wish to see them.

Recognizing that no one ever really creates a "polished" file from scratch, EZEDIT is included to allow you to get in and do your thing on your files.

Perhaps my favorite of all the programs was EZSORT. Almost limitless ways of sorting are allowed because of the wide variety of keys that can be used. Several keys can be used at once — one can sort by, say, Zip code, last name, and membership status if working with such a file. The program is intelligent: if sorting by last-name + first name + middle initial, the program can recognize that not all persons have middle initials. It runs right along without bombing in such circumstances. EZSORT performs a key sort using the keys that were established in EZCTL (EZCONTROL).

Only the disk addresses and keys are read, and only the disk addresses are rewritten. Work space and time are reduced by several orders of magnitude, using this approach. If the file to be sorted is too big to fit in memory, the program creates (and deletes) work files on disk without operator intervention. No time is wasted outputting files during or after sorts as all work is done directly on the disks.

EZPRNT will output to your printer or display on the terminal the results of your work. You may utilize the "Reduced Intensity" feature on your terminal in the display of protected fields and otherwise use every feature of your system throughout the group of programs, AS I UNDERSTAND THEM AT THIS TIME! I emphasize that because the programs are much too flexible to completely digest within a week or so.

Since files have a way of becoming obsolete, an EZPURG system is included to purge EZDATA files.

Anyone using extensive listings, such a mailing lists with keys, membership lists for organizations, subscription lists for publications, etc., should find it worthwhile to obtain and study this set of programs. It is already becoming a valuable addition to our file of software.

EZSORT alone would make this a worthwhile acquisition. But it is much, much more than just another sort system. The programs live up to their name of EZDATA MANAGEMENT SYSTEM. Anyone considering the system should, however, be aware that it is intended to work with an intelligent terminal. Direct cursor control is an absolute must.

The programs are f-a-s-t! In a telephone conversation, Gary mentioned that he runs his terminal at 9,600 baud. That gave me the idea of trying out the programs at various baud rates, since the SWTPCo terminal is software baud rate settable. They ran flawlessly from 50 baud to 38,400 baud. I could find no indication that the programs were slowing output at the highest rates.

I do wish, though, that the listings were more responsive to the paramters in the TTYSET of Flex 2.0. Some listings run through too quickly to digest and don't pause and can't be interrupted by the ESCAPE key. How about it, Gary? (I knew I could find at least one fault. I talk to Hixson so often on the phone that I've caught "Don Williamitis" — I always want one more feature!)

Comments to prospective users: beginner's should expect to sweat a while before making the programs run. Moderately skilled to skilled programmers should expect to spent a few minutes configuring the system and studying the supplied samples. I would rate this AAA material for anyone with moderate or better skills. It is serious materiel, not for game players. It is one of the better software programs we have received since we started computing. (And we have some goodies!)

For additional information contact: Gary A. Magnusen, 208 Tinkler, Lafayette, Indiana, 47901.

ED'S NOTE: Now how about a 6809 version Gary.

FAST PRIMES

Brian F. Bailey WB4MMP
5701 S.W. 4th St.
Plantation, FLA 33317

I read with interest the letter from Mr. Shirk of TSC comparing processor speeds (April 1981 issue, pp. 13-14). I agree completely that it is unfair to directly compare the speed of relocetable and/or reentrant programs with absolute code. I must take issue, however, with his comparison of the UNIFLEX Pascal program with the 6809 assembler version. I realize that the efficiency of the algorithm makes a large difference in timing. I therefore wish to show that assembly language is inherently faster than any high level language and that the 6809 is so superior to other 8 bit processors that there is really no comparison. The program PRIME, on this disk, takes advantage of the power of the 6809 so that it computes all of the prime numbers up to 10,000 in only about ONE SECOND (2 Mhz 68B09) or about two seconds at 1 Mhz. Try that on a Z-80! The memory required is N/2 bytes where N is the desired number limit. The primes are flagged in memory so that printout time (highly machine-dependent) is not counted as computation time. Please feel free to publish it if you want to. 73, Brian F. Bailey WB4MMP

```
PRIME NUMBER GENERATOR              6-6-4 TSC ASSEMBLER   PAGE   1
3/15/80   Brian F. Bailey

                  C003   WARMS   EQU   $C003    FLEX warmstart point
                  CD24   PCRLF   EQU   $CD24    Print CR/LF
                  CD1B   INBUFF  EQU   $CD1B    Input into line buffer
                  CD48   INDEC   EQU   $CD48    Input decimal number from line buffer
                  CD39   OUTDEC  EQU   $CD39    Output decimal number
                  CD1E   PDATA   EQU   $CD1E    Print text string
                  CC16   ESCRTN  EQU   $CC16    Escape return register
                         *
 0000 20   04            PRIMEB  BRA   PRIME    Bypass version number
 0002 01                 VERSN   FCB   1        Version one
 0003 7E   C003          DONE    JMP   WARMS
                         *
 0006 10CE 0000          PRIME   LDS   #PRIMEB  Initialize stack register
 000A 33   8D 00ED               LEAU  VARBSE,PCR Point to variable base address
 000E 8E   0006                  LDX   #PRIME   Point to beginning of program
 0011 BF   CC16                  STX   ESCRTN   Save in escape return pointer
 0014 30   8D 00CD               LEAX  LIMMSG,PCR Point to message
 0018 BD   CD1E                  JSR   PDATA    Print it
 001B BD   CD1B                  JSR   INBUFF   Input number into buffer
 001E BD   CD48                  JSR   INDEC    Get decimal number
 0021 27   E0                    BEQ   DONE     Branch if done (null entry)
 0023 AF   5E                    STX   -2,U     Save high limit
 0025 1F   10                    TFR   X,D      Move to D register
 0027 44                         LSRA           Divide MSB by two
 0028 56                         RORB           Divide LSB by two using carry (if any) from MSB
 0029 6F   C0            CLRLP   CLR   ,U+      Clear the Nth byte of the data area
 002B 83   0001                  SUBD  #1       Decrement the counter by one
 002E 26   F9                    BNE   CLRLP    Repeat until done
 0030 108E 0001                  LDY   #1       Preset initial Y value less two
 0034 31   22            LOOPA   LEAY  2,Y      Get next Y value
 0036 30   A4                    LEAX  ,Y       Load initial X value
 0038 CC   0000          MULTLP  LDD   #0       Clear product register
 003B 34   36                    PSHS  Y,X,D    Set up stack locations
 003D 34   06                    PSHS  D        Set up overflow word
 003F A6   65                    LDA   5,S      Get LSB of multiplicand
 0041 E6   67                    LDB   7,S      Get LSB of multiplier
 0043 3D                         MUL            Multiply
 0044 ED   62                    STD   2,S      Save partial product
 0046 A6   64                    LDA   4,S      Get MSB of multiplicand
 0048 E6   67                    LDB   7,S      Get LSB of multiplier
 004A 3D                         MUL            Multiply
 004B E3   61                    ADDD  1,S      Add partial product
 004D ED   61                    STD   1,S      Save partial product
 004F EC   65                    LDD   5,S      Get LSB of multiplicand and MSB of multiplier
 0051 3D                         MUL            Multiply
 0052 E3   61                    ADDD  1,S      Add partial product
 0054 ED   61                    STD   1,S      Save partial product
 0056 A6   64                    LDA   4,S      Get MSB of multiplicand
 0058 E6   66                    LDB   6,S      Get MSB of multiplier
 005A 3D                         MUL            Multiply
 005B E3   E1                    ADDD  ,S++     Add partial product and increment stack by two
 005D 26   14                    BNE   OVRFLW   Branch on overflow (over 16 bits)
```

```
PRIME NUMBER GENERATOR              4-4-4 TSC ASSEMBLER   PAGE   2
3/15/80   Brian F. Bailey

 0060 35   36          PULS  D,X,Y    Get product and restore multiplier and multiplicand
                    *
                    * NOTE:  The upper half of the 32 bit product is not
                    *        used since the upper limit is below 65536.
                    *
 0061 10A3 5E          CMPD  -2,U     Check product against upper bound
 0064 22   0F          BHI   NEXTY    Branch if higher
 0066 44               LSRA           Divide MSB of product by two
 0067 56               RORB           Divide LSB of product by two using carry (if any) from MSB
 0068 6A   CB  NTZERO  DEC   D,U      Decrement flag for this product
 006A 27   FC          BEQ   NTZERO   Make sure it is not decremented all the way to zero
 006C 30   02          LEAX  2,X      Get next X value
 006E 20   CB          BRA   MULTLP   Try next value
                    *
 0070 7E   CDC4  CRLF  JMP   PCRLF    Print CR/LF
                    *
 0073 35   36  OVRFLW  PULS  D,X,Y    Restore multiplicand and multiplier

 0075 34   20  NEXTY   PSHS  Y        Save Y on stack for comparison
 0077 AC   E1          CMPX  0,S++    Compare X to Y and restore stack
 0079 26   D9          BNE   LOOPA    Continue calculations if not equal
 007B 8D   F3          BSR   CRLF     Print CR/LF
 007D 8D   F1          BSR   CRLF     Print CR/LF
 007F 108E 0000        LDY   #0       Initialize prime counter
 0083 EC   5E          LDD   -2,U     Check high limit
 0085 1083 0002        CMPD  #2       Check against lowest prime
 0089 25   3A          BLO   NULL     End program if no primes
 008B CC   0002        LDD   #2       Get first prime pointer value
 008E 34   06  OUTLP   PSHS  D        Save prime pointer
 0090 44               LSRA           Divide MSB by two
 0091 56               RORB           Divide LSB by two, using carry (if any) from MSB
 0092 6D   CB          TST   D,U      Check prime flag for this pointer
 0094 26   0A          BNE   NOTPRM   Branch if not prime
 0096 31   21          LEAY  1,Y      Increment prime counter by one
 0098 30   E4          LEAX  0,S      Point to value on stack
 009A 5F               CLRB           Clear leading zero flag
 009B BD   CD39        JSR   OUTDEC   Print decimal number
 009E 8D   D0          BSR   CRLF     Print CR/LF
 00A0 35   06  NOTPRM  PULS  D        Restore prime pointer
 00A2 1083 0002        CMPD  #2       Is it the only even prime?
 00A6 26   03          BNE   ODD      Branch if not
 00A8 83   0001        SUBD  #1       Adjust pointer for subsequent add
 00AB C3   0002  ODD   ADDD  #2       Increment by two
 00AE 25   05          BCS   NULL     Stop on rollover
 00B0 10A3 5E          CMPD  -2,U     Compare to high limit
 00B3 23   D9          BLS   OUTLP    Repeat until done
 00B5 30   8D 0014 NULL LEAX PRMCNT,PCR Point to message
 00B9 BD   CD1E        JSR   PDATA    Print it
 00BC 34   20          PSHS  Y        Save prime counter on stack
 00BE 30   E4          LEAX  0,S      Point to it
 00C0 5F               CLRB           Clear leading zero flag
 00C1 BD   CD39        JSR   OUTDEC   Print decimal number
 00C4 35   20          PULS  Y        Restore stack pointer value

PRIME NUMBER GENERATOR              4-4-4 TSC ASSEMBLER   PAGE   3
3/15/80   Brian F. Bailey

 00C6 8D   A8          BSR   CRLF     Print CR/LF
 00C8 8D   A6          BSR   CRLF     Print CR/LF
 00CA 16   FF3F        LBRA  PRIME    Repeat
 00CD 0D 0A 00 00  PRMCNT FCC $0D,$0A,0,0,/Number of primes = /,4
 00E5 0D 0A 00 00  LIMERL FCC $0D,$0A,0,0,/Number limit? /,4
 00F9              *     RMB   2
           00FB VARSEL EQU   *        Prime flag storage = U/2
                       END   PRIMEN

0 ERROR(S) DETECTED

PRIME NUMBER GENERATOR              4-4-4 TSC ASSEMBLER   PAGE   4
3/15/80   Brian F. Bailey

SYMBOL TABLE:

CLRLP 0027  CRLF   0070  DONE   0003  ESCRIN CE1A  INBUFF CD1B
INDEC CD48  INPROG 00C5  LOOPA  0034  MULTLP 0030  N      0059
NEXTY 0075  NOTPRM 00A0  NTZERO 0068  NULL   00B5  ODD    00AB
OUTDEC CD39 OUTLP  008E  OVRFLW 0073  PCRLF  CDC4  PDATA  CD1E
PRIME 000A  PRMCNT 00CD  LIMERL 00E5  VARSEL 00FB  VERSN  0002
WARMS CD03
```

6809 JPC TC-3

TC-3 w/6809
by
Keith Alexander

Recently, a friend of mine bought a SwTPc 69/K computer, and like most of us started, he is limited to audio cassette tapes ("sloppy disks") for mass storage. We figured he could use my now-unused (since I now have a dual 5" disk system) tape hardware, a Southwest Tech. AC-30 or the JPC Products TC-3 tape interface (see ads in Micro Journal). This latter device had been the pinnacle of my tape storage when I got my disks. It had collected dust ever since then, though. For still-unknown reasons, we never got the AC-30 to work in his system, so we concentrated on the TC-3.

Now, let me tell you about this product. The interface, which resides on the 30-pin "peripheral" bus of the SwTPc chassis (which motherboard doesn't matter) comes not only with all parts, connectors, and a standard-sized (3.5" x 5.25"), solder-masked circuit board, but a comprehensive manual. The manual, which is beautifully written, gives circuit diagrams and commented assembly listings of all software to load and write both Kansas-City format (300 bits-per-second, or "baud") and "High Speed" format tapes, which may optionally be at 2400 or 4800 baud, depending on one or two "constant" values in the program. They also provide guidelines (names and model numbers) of several tape decks that have proven their suitability for the high-speed formats. Your $19.95 cheapie won't always do the trick. Mike and I have been using the Radio Shack SCT-12 (about $80), which finds other uses in the home audio system.

Then it hit me. I'd bought my TC-3 from JPC back in the Dark Ages (1978), for my (then) 6800. All the software was for 6800. I have dual processor capability built into my system, but Mike's is strictly 6809 (w/SBUG). No big problem really, just assemble it with TSC's 6809 Macro Assembler and the 6800 mnemonics will be assembled into 6809 code. Fine. This produced a 6809 program that I could load into memory off my disk, but which Mike would have to MANUALLY KEY IN WITH S-BUG'S MEMORY EXAMINE/CHANGE FUNCTION whenever he wanted to load a program into his computer. About 300 bytes. Now, I can tell you stories about when I used to routinely 'key-in' 2K and 3K 'object' listings, but now I cringe at the thought. No fun at all. Solution? Phase 2--put all the programs in ROM! Our 6809 cpu boards (SwTPc) can hold three 2716 EPROM memories (2K x 8 bit), besides SBUG. It doesn't make sense to put programs like this into a ROM in a 6809 system,

and not make them POSITION-INDEPENDENT. This enables them to be used not only in ANY of the three available PROM sockets, but anywhere in RAM, if you so desire. This was Phase 3.

Now, the fun begins. Since this was more or less a rush project, I didn't re-write the original JPC software. At first, I merely changed the necessary instructions to lend position independence (such as LEAX MSG,PCR instead of LDX #MSG). Like the original software, I still used a designated area of RAM for variable storage, some of which had to be PRESET (with address ranges) before running three of the four programs. The next revision took an idea proposed by Mr. Al Moreira of England (see 7-80 Micro Journal, p.31)-- all local variables and temporary storage is allocated in the AREA POINTED TO BY THE STACK REGISTERS, US & SP. The stack is bumped down upon program entry, and restored when exited. The final revision makes the programs "friendlier", by prompting the user for the start and end addresses of the ranges to be loaded or dumped. This was mostly a matter of using the IN2ADR routine in SBUG, and adding some range-checking.

In summary, the following four programs (KC Loader & Writer, and High-Speed Loader & Writer) are:

ROM-ABLE (use no RAM outside of stack)
POSITION-INDEPENDENT
PROMPTING (no need to preset anything)
USE SOME SBUG ROUTINES
PRESUME A 1 MHZ. SYSTEM CLOCK

One other thing. This little project taught me all about the new I/O port addressing scheme used on SwTPc's newer motherboards. My original 6800 software and motherboard used 4-address ports, that is, I/O port #2 is $8008-800B. My kluged motherboard (that uses 6800 or 6809 cpu boards and can address I/O at $8000 or $E000) still uses four locations per port, while Mike's newer (standard) board uses SIXTEEN memory locations for each I/O slot. If we both use I/O port #0, our software could be identical, but after port 0, our port addresses are all different. The 6821 on the TC-3 board occupies the LOWEST FOUR (at least) addresses on whatever I/O slot you use. The point is that you may have to adjust the port address references to suit your particular system, depending on which slot you put the TC-3 on, and what kind of addressing scheme your I/O bus has.

As I pointed out, the programs do use routines in SBUG, mostly for printing strings and getting the required input from the user. These restraints shouldn't prove too formidable for someone using a different monitor, though. Just a matter of plugging in addresses, I would think.

The observant user will notice I retained some of the 6800 mnemonics from the original software, and changed others. In the KC Loader, for instance, note that I retained 'TAB' instead of replacing it with the 6809's equivalent TFR A,B. However, note also that the TSC 6809 Macro Assembler also translated TAB to the equivalent SEQUENCE TFR A,B, TST A (1F89, 4D), which is nice.

It should be pointed out that the 300-baud Kansas City Standard was only meant as a standard for EXCHANGE of software BETWEEN machines, not necessarily for MASS STORAGE on ONE machine. The more enterprising bit-twiddler with a 6809 tape system will find this hardware/software system valuable not only for the sake of being able to dump and load tapes in two different formats, but for such bigger programs as BASIC interpreters, compilers, Editors and Assemblers (that are themselves tape-interactive) with a little horsing around and careful patching, you could speed THEIR 'save' and 'load' functions up possibly eightfold or more, providing you can find the original SAVE and LOAD entry points-- not always so easy.

```
        NAM     HSL2400.ASM

        TTL     2400 BAUD TC-3 LOADER
        STTL    IN 6809 P.I.C.

* October 15, 1980
* (rev. 2-8-81)
* (rev. 3-12-81)
*
* High speed (to 2400 baud) Tape LOAD Utility for JPC
* Products, Inc. TC-3 Audio Cassette Interface
*
* Written to reside in PROM and use the STACK AREA for VARIABLE
* STORAGE. This makes it absolutely position-independent.
*

        OPT     PAG
```

```
        *   I/O PORT ADDRESSES FOR TC-3 (ALTER TO SUIT)

E000    PRTA    EQU     $E000       PIA DATA REG. ADDR.
E009    CTLA    EQU     $E009       PIA CONTROL REG. ADDR.

        *       EXTERNAL EQUATES (SBUG MONITOR)

F814    BYE     EQU     $F814       MONITOR RE-ENTRY POINT (ALTER TO SUIT)
FDAE    PSTRNG  EQU     $FDAE       PRINT STRING POINTED AT BY XR 'TIL EOT (04)
FD20    IN2ADR  EQU     $FD20       GET TWO ADDRS. FROM USER (TO YR,XR)

        *    ASCII EQUATES

000A    LF      EQU     $A
000D    CR      EQU     $D
0004    EOT     EQU     4
0007    BEL     EQU     7

        *       STACK AREA VARIABLE STORAGE
        * Values equal offsets from user stack (US)

0000    NUMB    EQU     0           Bit count (1 byte)
0001    SREF    EQU     1           Signal reference state (1 byte)
0002    CSUM    EQU     2           Eight-bit checksum
0003    BEGA    EQU     3           Beginning address (2 bytes)
0005    ENDA    EQU     5           Ending address
0007    SIZE    EQU     7           NUMBER OF BYTES RESERVED FOR LOCAL VARIABLES

        * NO ORG SPECIFIED; DEFAULTS TO $0000. CAN BE ORG'D ANYWHERE.

0000 20   01        BEGIN   BRA     UTL0

0002 04             VN      FCB     4           (3-12-81)

0003 32   79        UTL0    LEAS    -SIZE,S     Bump 'SYS' stack down by req'd amount so
0005 1F   4                 TFR     S,U         subrouts. won't interfere w/variable storage
0007 30   8D 008C           LEAX    MSG1,PCR
000B BD   FDAE              JSR     PSTRNG
000E BD   FD20              JSR     IN2ADR      Input address range to YR, XR. no checking.
0011 10AF 43                STY     BEGA,U      Hmm.
0014 AF   45                STX     ENDA,U      Huh ?
0016 10AC 45                CMPY    ENDA,U      Check for invalid range...
0019 25   04                BCS     START       O.K. if Beg. < End
001B 32   67                LEAS    SIZE,S      Else, fix up stack...
001D 20   E1                BRA     BEGIN       and restart.
        *
        *
001F 30   8D 00AF   START   LEAX    MSG2,PCR    Range OK. BEGIN LOAD HERE
0023 BD   FDAE              JSR     PSTRNG
        *
        * FROM HERE ON CODE IS VIRTUALLY THE SAME AS JPC PRODUCTS, INC.
        * ORIGINAL. ONLY ADDRESSING MODES HAVE BEEN CHANGED TO LEND
        * POSITION-INDEPENDENCE, IMPLEMENT STACK VARIABLE
        * STORAGE, AND PROTECT THE INNOCENT.
        *
0026 86   04                LDAA    #4
0028 B7   E009              STAA    CTLA        Enable PIA Port A
002B 86   FF        UTL1    LDAA    #$FF        Wait for solid leader
002D 8D   54        UTL2    BSR     EDGE        Test for presence of leader
002F 2B   FA                BMI     UTL1        If not leader, start over
0031 4A                     DECA                Otherwise decrement counter
0032 26   F9                BNE     UTL2        Until time is up
0034 8D   2D        UTL3    BSR     LODR        Get first byte
0036 81   03                CMPA    #$03        Must be BOF...
0038 26   FA                BNE     UTL3        ...or keep looking

003A 6F   42                CLR     CSUM,U      Zero out checksum
003C AE   43                LDX     BEGA,U      Set up memory pointer
003E 30   1F                DEX                 Adjust pointer

0040 30   01        UTL4    LEAX    1,X         Increment pointer
0042 8D   1F                BSR     LODR        Get byte from tape
0044 A7   84                STAA    0,X         Store in memory
0046 AB   42                ADDA    CSUM,U      Add byte into checksum
0048 A7   42                STAA    CSUM,U      ...and save it
004A AC   45                CMPX    ENDA,U      Done yet?
004C 26   F2                BNE     UTL4        If not keep loading

004E 8D   13                BSR     LODR        Check byte from tape
0050 A1   42                CMPA    CSUM,U      Compare with running checksum
0052 26   0E                BNE     LERR        If not same report load error

0054 30   8D 0075           LEAX    BELL,PCR    Else, ring bell...
0058 BD   FDAE              JSR     PSTRNG      ...to signal finish
005B 1F   34                TFR     U,S         Guarantee clean exit...
005D 32   67                LEAS    SIZE,S      by fixing up SP...
005F 7E   F814              JMP     BYE         and exit back to monitor (or wherever)

0062 3F             LERR    SWI                 *** Software interrupt here on load error ***

        *
        * Loader Subroutine
        *
0063 C6   09        LODR    LDAB    #9          Set bit count
0065 E7   C4                STAB    NUMB,U
0067 8D   1A                BSR     EDGE        Presynchronize
0069 8D   18        LOD1    BSR     EDGE        Read next bit
006B 2A   FC                BPL     LOD1        Loop and wait for start bit
006D 8D   14                BSR     EDGE        Throw away second half of 1 bit
006F 6A   C4        LOD2    DEC     NUMB,U      Decrement bit count
0071 2D   0F                BLT     LOD4        Exit if all 8 bits done
0073 44                     LSRA                Shift the data right 1 bit
0074 8D   0D                BSR     EDGE        Get next input bit
0076 2A   04                BPL     LOD3        If bit = 0 that's it...
0078 8D   09                BSR     EDGE        ...or if it's a 1, resynch
007A 8A   80                ORAA    #$80        Push new bit into data word
007C 20   F1                BRA     LOD2
007E 8A   00        LOD3    ORAA    #0          Just to equalize timing
0080 20   ED                BRA     LOD2

0082 39             LOD4    RTS                 Return to main program

0083 34   02        EDGE    PSHA                Save data in A-register
0085 4F                     CLRA                Clear count
0086 4C             EDG2    INCA                Increment count
0087 F6   E000              LDAB    PRTA        Test input port A
008A C4   01                ANDB    #1
008C E1   41                CMPB    SREF,U      Input same as reference? (last look)
008E 27   F6                BEQ     EDG2        If so, wait for change

0090 E7   41                STAB    SREF,U      save for reference
        ****************
0092 80   10                SUBA    #$10        THIS CONSTANT DETERMINES BAUD RATE
        *                                           *** $10 FOR 2400B., $09 FOR 4800B. ***
        ****************
0094 35   02                PULA                Restore A Register
0096 39                     RTS                 Return
        *
```

```
        * STRINGS
        *
0097 0A 0D 2A 2A    MSG1    FCC     LF,CR,'** High Speed TC-3 Tape Loader **',LF,LF,CR
009B 20 48 69 67
009F 68 20 53 70
00A3 65 65 64 20
00A7 54 43 2D 33
00AB 20 54 61 70
00AF 65 20 4C 6F
00B3 61 64 65 72
```

```
2400 BAUD TC-3 LOADER                   3-12-81 TSC 6809 ASMB.  PAGE   3
IN 6809 P.I.C.

00B7 2A 2A 2A 0A
00BB 0A 0D
00BD 52 61 6E 67            FCC     "Range to be loaded? ",EOT
00C1 65 20 74 6F
00C5 20 62 65 20
00C9 6C 6F 61 64
00CD 65 64 3F 20
00D1 04

00D2 0A 0D 0A 4F    MSG2    FCC     LF,CR,LF,'OK. Load in progress...',EOT
00D6 4B 2E 20 4C
00DA 6F 61 64 20
00DE 69 6E 20 70
00E2 72 6F 67 72
00E6 65 73 73 2E
00EA 2E 2E 04

00ED 07 00 00 04    BELL    FCC     BEL,0,0,EOT

                            END     BEGIN

0 ERROR(S) DETECTED
```

```
        * October 29, 1980
        * (rev. 2-25-81)
        *
                NAM     KCLPIC.ASM

                TTL     6809 KC LOADER FOR TC-3

        *
        * Kansas City Standard LOADER for JPC TC-3 ACR interface
        *
                OPT     PAG

        * I/O PORT ADDRESSES FOR TC-3 (ALTER TO SUIT)

E000    PRTA    EQU     $E000       I/O Port #2 on 'paleolithic' (MP-B) motherboard
E009    CTLA    EQU     $E009

        *    EXTERNAL EQUATES (SBUG MONITOR)

F814    BYE     EQU     $F814       MONITOR RE-ENTRY POINT
FDAE    PSTRNG  EQU     $FDAE
FDC8    INCH    EQU     $FDC8       INPUT CHAR. TO ACC. A

        *    ASCII EQUATES

000A    LF      EQU     $A
000D    CR      EQU     $D
0007    BEL     EQU     7
0004    EOT     EQU     4

        *    STACK AREA VARIABLE STORAGE
        * variables expressed as offsets from stack (US).

0000    XHIA    EQU     0           Address pointer - MSB (1 byte)
0001    XLOA    EQU     1           Address pointer - LSB (1 byte)
0002    CKSM    EQU     2           Checksum (1 byte)
0003    SREF    EQU     3           Reference state (1 byte)
0004    XTEM    EQU     4           Temporary XREG (2 bytes)
0006    NUMB    EQU     6           Bit count
0007    BYTC    EQU     7           Temporary byte count (1 byte)
0008    SIZE    EQU     8           NUMBER OF BYTES RESERVED ON STACK (US)

        * NO ORG SPECIFIED; CAN BE ANYWHERE IN MEMORY

0000 20   01        BEGIN   BRA     LOAD

0002 05             VN      FCB     5           (2-25-81)

0003 32   78        LOAD    LEAS    -SIZE,S     Set up variable stg. area on stack
0005 1F   43                TFR     S,U         US REMAINS UNCHANGED FROM HERE ON (SP-SIZE)
0007 30   8D 00EA           LEAX    MSG1,PCR
000B BD   FDAE              JSR     PSTRNG
000E BD   FDC8              JSR     INCH
0011 81   0D                CMPA    #$D         Carriage "RETURN" input?
0013 27   04                BEQ     LOAD1

0015 32   68                LEAS    SIZE,S
0017 20   EA                BRA     LOAD
0019 30   8D 0138   LOAD1   LEAX    MSG2,PCR
001D BD   FDAE              JSR     PSTRNG
0020 7F   E009              CLR     CTLA        Set up PIA
0023 86   02                LDAA    #2          Set data direction
0025 B7   E000              STAA    PRTA
0028 86   04                LDAA    #4          Enable port
002A B7   E009              STAA    CTLA

002D 8D   71        LOA1    BSR     KCIN        Read first character
002F 81   53                CMPA    #'S         If not an 'S'...
0031 26   FA                BNE     LOA1        Then continue search

0033 8D   6B                BSR     KCIN        Read second character
0035 81   39                CMPA    #'9         If a '9' then...
0037 26   0E                BNE     LOA2

0039 30   8D 0133           LEAX    BELL,PCR    Signal finished by
003D BD   FDAE              JSR     PSTRNG      by ringing bell

0040 1F   34                TFR     U,S         Guarantee clean exit
0042 32   68                LEAS    SIZE,S      stack-wise (SP)
0044 7E   F814              JMP     BYE         and return to SBUG (or wherever)

0047 81   31        LOA2    CMPA    #'1         If not a '1'...
0049 26   E2                BNE     LOA1        ...then try again.
004B 6F   42                CLR     CKSM,U      Clear checksum
004D 8D   24                BSR     BYTE        Read byte count in hex
004F 80   02                SUBA    #2          Adjust for address bytes and...
0051 A7   47                STAA    BYTC,U      ...store
```

```
0053 8D    1E                BSR    BYTE      Read start address (high)...
0055 A7    C4                STA    XHIA,U    ...and store
0057 8D    1A                BSR    BYTE      Read start address (low)...
0059 A7    41                STA    XLOA,U    ...and store

005B AE    C4                LDX    XHIA,U    Initialize address pointer

005D 8D    14         LOA3   BSR    BYTE      Get data

005F 6A    47                DEC    BYTC,U    Adjust byte count
0061 27    0A                BEQ    LOA4      If last byte - verfy checksum

0063 A7    84                STA    0,X       Otherwise store data and...
0065 A1    84                CMPA   0,X       ...verify correct storage
0067 26    09                BNE    LOA6      If not - error

0069 30    01                INX              Update address pointer...
006B 20    F0                BRA    LOA3      ...and continue

006D 6C    42         LOA4   INC    CKSM,U    Verify checksum
006F 27    0C                BEQ    LOA1

0071 3F               LOA5   SWI              ***Checksum Error***
0072 3F               LOA6   SWI              ***Memory Storage Error***
                      *
                      * Byte Input Subroutine
                      *
0073 8D    15         BYTE   BSR    INHX      Get hex value - high
0075 48                      ASLA             Adjust data to high order nibble
0076 48                      ASLA
0077 48                      ASLA
0078 48                      ASLA

0079 1F    894D              TAB              Preserve in B-Register

007C 8D    0C                BSR    INHX      Get hex value - low

007E 34    04 ABE0           ABA              Combine high and low nibbles

0082 1F    894D              TAB              Update checksum
0085 EB    42                ADDB   CKSM,U
0087 E7    42                STB    CKSM,U

0089 39                      RTS              Return
                      *
                      * Read Hex Subroutine
                      *
008A 8D    34         INHX   BSR    KCIN      Read one ASCII character
008C 80    30                SUBA   #$30      Convert to hex
008E 2B    0F                BMI    INH2      Not hex-ASCII if results < 0

0090 81    09                CMPA   #09       Determine if hex-ASCII (A-F)
0092 2F    0A                BLE    INH1

0094 81    11                CMPA   #$11
0096 2B    07                BMI    INH2

0098 81    16                CMPA   #$16
009A 2E    03                BGT    INH2

009C 80    07                SUBA   #7        Yes - adjust value

009E 39               INH1   RTS              Return
009F 3F               INH2   SWI              ***Hex-ASCII Error***
                      *
                      * Input one byte subroutine
                      *
00A0 34    04         KCIN   PSHB             Preserve the B-register

00A2 8D    2B         KCI1   BSR    EDGE      Get Start bit
00A4 2D    FC                BLT    KCI1
00A6 8D    27                BSR    EDGE      Move down bit stream
00A8 8D    25                BSR    EDGE      If false start...
00AA 2D    F6                BLT    KCI1      ...try again

00AC C6    08                LDB    #8        Initialize bit count and...
00AE E7    46                STB    NUMB,U
00B0 4F                      CLRA             ...data value register

00B1 8D    10                BSR    DELY      Delay one bit time
00B3 44               KCI2   LSRA             Shift for next bit
00B4 8D    2D                BSR    STAT      What is bit status?
00B6 2E    02                BGT    KCI3      If '0' - continue
00B8 8A    80                ORA    #$80      '1' - set bit

00BA 8D    07         KCI3   BSR    DELY      Delay one bit time

00BC 6A    46                DEC    NUMB,U    If not finished...
00BE 2E    F3                BGT    KCI2      ...get next bit.

00C0 35    04                PULB             Restore the B-register
00C2 39                      RTS              Return
                      *
                      * Delay Subroutine
                      *
00C3 AF    44         DELY   STX    XTEM,U    Preserve X-register
00C5 8E    0150              LDX    #$0150    ***NOTE: CONSTANT DETERMINES BAUD RATE ***

00C8 30    1F         DEL1   DEX
00CA 26    FC                BNE    DEL1
00CC AE    44                LDX    XTEM,U    Restore X-register
00CE 39                      RTS              Return
                      *
                      * Frequency Counter Subroutine
                      *
00CF 34    02         EDGE   PSHA             Save data in A-register
00D1 4F                      CLRA             Clear count
00D2 4C               EDG2   INCA             Increment count
00D3 F6    E000              LDB    PRTA      Test input port A
00D6 C4    01                ANDB   #1
00D8 E1    43                CMPB   SREF,U    If same as reference...
00DA 27    F6                BEQ    EDG2      ...wait for change
00DC E7    43                STB    SREF,U    Save for reference
00DE 80    10                SUBA   #$10      ***NOTE: CONSTANT DETERMINES FREQUENCY ***
00E0 35    02                PULA             Restore A-register
00E2 39                      RTS              Return
                      *
                      * Bit Status Subroutine
                      *
00E3 F6    E000       STAT   LDB    PRTA      Set up SREF with current...
00E6 C4    01                ANDB   #1        ...port value
00E8 E7    43                STB    SREF,U
00EA 8D    E3                BSR    EDGE      Determine wave form
00EC 8D    E1                BSR    EDGE
00EE 2E    04                BGT    STA1
00F0 8D    DD                BSR    EDGE      '1' timing adjustment
00F2 8D    DB                BSR    EDGE
00F4 39               STA1   RTS              Return
                      *
```

Listing Continued P-21

```
                             NAM    HSWPIC.ASM

                             TTL    2400-BAUD TC-3 WRITER
                             STTL   IN 6809 P.I.C.
                      *
                      * October 13, 1980
                      * (rev.2-25-81)
                      * (rev.3-12-81)
                      *
                      * High speed (to 2400 baud) Tape WRITE Utility for JPC
                      * Products, Inc. TC-3 Audio Cassette Interface.
                      *
                             OPT    PAG

                      * I/O PORT ADDRESSES FOR TC-3 (ALTER TO SUIT)

          E000 PRTA          EQU    $E000     PIA DATA REG.ADDR.
          E001 CTLA          EQU    $E001     PIA CONTROL REG.ADDR.

                      *   EXTERNAL EQUATES (IN SBUG MONITOR)

          F814 BYE           EQU    $F814     MONITOR RE-ENTRY POINT (ALTER TO SUIT)
          FD4E PSTRNG        EQU    $FD4E     PRINT STRING POINTED AT BY XR 'TIL EOT (04)
          FD20 IN2ADR        EQU    $FD20     GET TWO ADDRS. FROM USER (TO YR,XR)

                      *      ASCII EQUATES

          000A LF            EQU    $A
          000D CR            EQU    $D
          0004 EOT           EQU    4
          0007 BEL           EQU    7

                      *    STACK AREA VARIABLE STORAGE
                      *  values equal offsets from user stack (US)

          0000 NUMB          EQU    0         bit count (1 byte)
          0001 CSUM          EQU    1         Eight-bit checksum (1 byte)
          0002 BEGA          EQU    2         Beginning address (2 bytes)
          0004 ENDA          EQU    4         Ending address (2 bytes)
          0006 SIZE          EQU    6         QTY OF BYTES RESERVED ON STACK

                      * NO ORG SPECIFIED; CAN BE ORG'D ANYWHERE.

0000 20    01         BEGIN  BRA    UTWR

0002 03               VN     FCB    3         (3-12-81)

0003 32    7A         UTWR   LEAS   -SIZE,S   Set up variable stg. area on stack
0005 1F    43                TFR    S,U
0007 30    8D 0080           LEAX   MSG1,PCR
000B BD    FD4E              JSR    PSTRNG
000E BD    FD20              JSR    IN2ADR    Input address range to YR,XR
0011 10AF  42                STY    BEGA,U

0014 AF    44                STX    ENDA,U
0016 10AC  44                CMPY   ENDA,U    Check for invalid range...
0019 25    04                BCS    START     O.K. if Beg. < End
001B 32    66                LEAS   SIZE,S    Else, fix up stack
001D 20    E1                BRA    BEGIN     and restart
                      *
                      *
001F 30    8D 00EE    START  LEAX   MSG2,PCR
0023 BD    FD4E              JSR    PSTRNG
0026 7F    E001              CLR    CTLA      Set up DDR
0029 86    02                LDA    #2
002B B7    E000              STA    PRTA
002E 86    04                LDA    #4        Enable Port PIA
0030 B7    E001              STA    CTLA

0033 8E    3FFF              LDX    #$3FFF    Write leader
0036 8D    30         UTW1   BSR    WLED
0038 30    1F                DEX
003A 26    FA                BNE    UTW1      Loop until zero

003C 86    03                LDA    #$03      Write 'BOF' (Beginning of File) char.
003E 8D    2C                BSR    WRTR

0040 6F    41                CLR    CSUM,U    Zero out checksum
0042 AE    42                LDX    BEGA,U    Set memory pointer
0044 30    1F                DEX              Adjust pointer

0046 30    01         UTW2   INX              Increment pointer
0048 A6    84                LDA    0,X       Get byte
004A 8D    20                BSR    WRTR      Write byte on tape
004C A6    84                LDA    0,X       Recover byte...
004E AB    41                ADDA   CSUM,U    Add it into checksum...
0050 A7    41                STA    CSUM,U    And save checksum
0052 AC    44                CPX    ENDA,U    Done yet?
0054 26    F0                BNE    UTW2      If not, write next byte

0056 A6    41                LDA    CSUM,U    Else, put checksum on tape
0058 8D    12                BSR    WRTR

005A 30    8D 00CE           LEAX   BELL,PCR  Signal finish...
005E BD    FD4E              JSR    PSTRNG    ...by ringing bell
0061 1F    34                TFR    U,S       Clean up SP...
0063 32    66                LEAS   SIZE,S    For graceful...
0065 7E    F814              JMP    BYE       ...exit.
                      *
                      * Write Subroutine
                      *
0068 34    04         WLED   PSHB             Save B register
006A 20    10                BRA    WRT2      Create leader
006C 34    04         WRTR   PSHB             Save B register
006E C6    08                LDB    #8        Set bit count = 8
0070 E7    C4                STB    NUMB,U

0072 8D    1C                BSR    CORD      Create start bit
0074 8D    1A                BSR    CORD

0076 8D    18         WRT1   BSR    CORD      Create synch
0078 85    01                BITA   #1        Test the data bit
007A 27    04                BEQ    *+6
007C 8D    12                BSR    CORD      Complement output for a one...
007E 20    02                BRA    *+4
0080 8D    17                BSR    NOCO      ...but not for a zero

0082 6A    C4                DEC    NUMB,U    Decrement bit count
0084 2F    03                BLE    WRT2      If 8 bits done, exit
0086 44                      LSRA             Shift data right for next bit
0087 20    ED                BRA    WRT1      Do next bit

0089 8D    05         WRT2   BSR    CORD      Create stop bit
008B 8D    0C                BSR    NOCO
008D 35    04                PULB             Restore B register
008F 39                      RTS              Return
                      *
                      * Create bit stream on tape
                      *
0090 F6    E000       CORD   LDB    PRTA      Test output port
0093 C5    02                BITB   #2
0095 26    04                BNE    CLRO      If set...clear it
0097 20    0C                BRA    SETO      If clear...set it
```

```
0099 20    1A        HOCO   BRA    DLY2         Do double delay for zeroes
009B F6    E008      CLRO   LDAB   PRTA         Clear output bit
009E C4    FD               ANDB   #$FD
00A0 F7    E008             STAB   PRTA
00A3 20    0A               BRA    DLAY         Do single delay
00A5 F6    E008      SETO   LDAB   PRTA         Set output bit
00A8 CA    02               ORAB   #2
00AA F7    E008             STAB   PRTA
00AD 20    00               BRA    DLAY         Just to equalize time
                     ******************
00AF C6    12        DLAY   LDAB   #$12         *** THIS CONSTANT DETERMINES BAUD RATE ***
                     *                              *** USE $12 FOR 2400b., $09 FOR 4800b. ***
                     ******************
00B1 5A                     DECB
00B2 26    FD               BNE    DLAY+2       Loop until zero
00B4 39                     RTS                 Return

                     ******************
00B5 C6    26        DLY2   LDAB   #$26         *** THIS CONSTANT DETERMINES BAUD RATE ***
                     *                              *** USE $26 FOR 2400b., $16 FOR 4800b. ***
                     ******************
00B7 5A                     DECB
00B8 26    FD               BNE    DLY2+2       Loop until zero
00BA 39                     RTS                 Return
                     *
                     *   STRINGS
```

```
2400-BAUD TC-3 WRITER                3-12-81 TSC 6809 ASMB.  PAGE    3
IN 6809 P.I.C.

                     *
00BB 0A 0D 2A 2A     MSG1   FCC    LF,CR,"** High Speed TC-3 Tape Writer **",LF,LF,CR
00BF 20 48 69 67
00C3 68 20 53 70
00C7 65 65 64 20
00CB 54 43 2D 33
00CF 20 54 61 70
00D3 65 20 57 72
00D7 69 74 65 72
00DB 20 2A 2A 0A
00DF 0A 0D
00E1 53 54 41 52            FCC    "START RECORDER, and input range to be written: ",EOT
00E5 54 20 52 45
00E9 43 4F 52 44
00ED 45 52 2C 20
00F1 61 6E 64 20
00F5 69 6E 70 75
00F9 74 20 72 61
00FD 6E 67 65 20
0101 74 6F 20 62
0105 65 20 77 72
0109 69 74 74 65
010D 6E 3A 20 04

0111 0A 0D 0A 4F     MSG2   FCC    LF,CR,LF,"OK, Dump in progress...",EOT
0115 4B 2E 20 44
0119 75 6D 70 20
011D 69 6E 20 70
0121 72 6F 67 72
0125 65 73 73 2E
0129 2E 2E 04

012C 07 00 00 04     BELL   FCC    BEL,0,0,EOT

                            END    BEGIN

0 ERROR(S) DETECTED
```

```
                            NAM    KCWPIC.ASM

                            TTL    6809 KC WRITER for TC-3

                            OPT    PAG

                     *
                     * October 12, 1980
                     * (rev.3-12-81)
                     *
                     * Kansas City Standard WRITER for JPC TC-3 ACR Interface
                     *
                     *
                     * I/O PORT ADDRESSES FOR TC-3

           E008 PRTA  EQU  $E008   PIA DATA REG.ADDR.
           E009 CTLA  EQU  $E009   PIA CONTROL REG.ADDR.

                     * EXTERNAL EQUATES (SBUG ROUTINES)

           F814 BVE     EQU  $F814   SBUG RE-ENTRY POINT
           FD4E PSTRNG  EQU  $FD4E
           FD20 IN2ADR  EQU  $FD20

                     * ASCII EQUATES

           000A LF    EQU  $A
           000D CR    EQU  $D
           0007 BEL   EQU  7
           0004 EOT   EQU  4

                     * STACK AREA VARIABLE STORAGE
                     * variables expressed as offsets from stack (US)

           0000 BEGA  EQU  0       Beginning address
           0002 ENDA  EQU  2       Ending address
           0004 TEMP  EQU  4       Temporary storage address
           0005 DELT  EQU  5       Temporary storage address
           0006 XTEM  EQU  6       Temporary X-register
           0008 NUMB  EQU  8       Bit count
           0009 BYTC  EQU  9       Temporary byte count
           000A SIZE  EQU  10      NUMBER OF RESERVED BYTES

                     * NO ORG SPECIFIED; WILL WORK ANYWHERE IN MEMORY

0000 20    01        BEGIN  BRA    KCWR

0002 03              VN     FCB    3            (3-12-81)

0003 32    76        KCWR   LEAS   -SIZE,S
0005 1F    43               TFR    S,U          US REMAINS UNCHANGED FROM HERE ON (SP-SIZE)
0007 30    8D 0119          LEAX   MSG1,PCR
000B BD    FD4E             JSR    PSTRNG
000E BD    FC20             JSR    IN2ADR       Input address range
0011 10AF  C4               STY    BEGA,U
0014 AF    42               STX    ENDA,U
0016 10AC  42               CMPY   ENDA,U
0019 25    04               BCS    START        O.K. if Beg < End
001B 32    6A               LEAS   SIZE,S       Else restore SP
001D 20    E1               BRA    BEGIN        and restart

001F 30    8D 015F   START  LEAX   MSG2,PCR
0023 BD    FD4E             JSR    PSTRNG
0026 8D    75               BSR    WLED         Initialize PIA and write leader
0028 86    02               LDA    #2
002A BA    E008             ORA    PRTA         Leave leader in 'high' state
002D B7    E008             STA    PRTA
0030 AE    C4               LDX    BEGA,U
0032 AF    46               STX    XTEM,U
0034 EC    42        KCW1   LDD    ENDA,U       Calculate record byte count
0036 A3    46               SUBD   XTEM,U
0038 1083  0010             CMPD   #$10         If greater than 16...
003C 25    02               BCS    KCW3
003E 86    0F        KCW2   LDA    #$F          ...force 16.

0040 8B    04        KCW3   ADDA   #4           Adjust count for address bytes
0042 A7    49               STA    BYTC,U
0044 80    03               SUBA   #3
0046 A7    48               STA    NUMB,U
0048 86    0D               LDA    #$0D         Generate CR
004A 17    0097             LBSR   WBYT
004D 86    0A               LDA    #$0A         Generate LF
004F 17    0092             LBSR   WBYT
0052 86    53               LDA    #'S          Generate "S"
0054 17    008D             LBSR   WBYT
0057 86    31               LDA    #'1          Generate "1"
0059 17    0088             LBSR   WBYT
005C 5F                     CLRB                Clear checksum
005D 30    49               LEAX   BYTC,U       Output record byte count
005F 17    0083             LBSR   KCBT

0062 30    46               LEAX   XTEM,U       Output record start address
0064 17    007E             LBSR   KCBT
0067 17    0078             LBSR   KCBT

006A AE    46               LDX    XTEM,U
006C 17    0076      KCW4   LBSR   KCBT         Output record bytes...
006F 6A    48               DEC    NUMB,U
0071 26    F9               BNE    KCW4         ...till record complete

0073 AF    46               STX    XTEM,U       Preserve next address
0075 53                     COMB                Calculate and output...
0076 34    04               PSHS   B
0078 1F    41               TFR    S,X
007A 17    0000             LBSR   KCBT         ...checksum
007D 35    04               PULS   B

007F AE    46               LDX    XTEM,U       Determine if all data...
0081 30    1F               LEAX   -1,X         ...has been output
0083 AC    42               CMPX   ENDA,U       If not...
0085 26    AD               BNE    KCW1         ...continue

0087 86    53               LDA    #'S          All done...
0089 8D    39               BSR    WBYT         Output "S9"
008B 86    39               LDA    #'9
008D 8D    35               BSR    WBYT

008F 30    8D 0099          LEAX   BELL,PCR     Signal finished
0093 BD    FD4E             JSR    PSTRNG       by ringing bell

0096 1F    34               TFR    U,S          Guarantee a tidy
0098 32    6A               LEAS   SIZE,S       exit, stack-wise
009A 7E    F814             JMP    BVE          and return to SBUG (OR WHEREVER)
                     *
                     * PIA Initialization and Leader Subroutine
                     *
009D 7F    E009      WLED   CLR    CTLA         Initialize PIA
00A0 86    02               LDA    #2           Set data direction
00A2 B7    E008             STA    PRTA
00A5 86    04               LDA    #4           Enable port
00A7 B7    E009             STA    CTLA

00AA 8E    06FF      WLE1   LDX    #$6FF        Generate leader
00AD 8D    0D        WLE2   BSR    WBIW         "1"
00AF 30    1F               LEAX   -1,X
00B1 26    FA               BNE    WLE2
00B3 39                     RTS                 Return
                     *
                     * Data Bit Output Subroutine
                     *
00B4 34    06        WBIZ   PSHS   A,B          Entry for '0' data bit output
00B6 C6    04               LDB    #4           1200 Hz. Baud count

00B8 86    30               LDA    #$30         *** NOTE: THIS CONSTANT DETERMINES FREQUENCY ***

00BA 20    06               BRA    WBI1

00BC 34    06        WBIW   PSHS   A,B          Entry for '1' data bit output
00BE C6    08               LDB    #8           2400 Hz. Baud count

00C0 86    25               LDA    #$25         *** NOTE: THIS CONSTANT DETERMINES FREQUENCY ***

00C2 A7    43        WBI1   STA    DELT,U       Preserve delay time
00C4 86    02        WBI2   LDA    #2           Generate square wave
00C6 BA    E008             ORA    PRTA         ...With frequency...
00C9 B7    E008             STA    PRTA         ...equal to DELT.
00CC A6    45               LDA    DELT,U

00CE 4A              WBI3   DECA                Delay
00CF 26    FD               BNE    WBI3
00D1 86    02               LDA    #2           Complement for square wave
00D3 B8    E008             EORA   PRTA
00D6 B7    E008             STA    PRTA
00D9 A6    45               LDA    DELT,U
00DB 4A              WBI4   DECA                Delay
00DC 26    FD               BNE    WBI4
00DE 5A                     DECB                If not complete...
00DF 26    E3               BNE    WBI2         ...continue

00E1 35    06               PULS   A,B          Restore registers A and B
00E3 39                     RTS                 Return
                     *
                     * Write Byte Subroutine
                     *
00E4 34    04        WBYT   PSHS   B            Output 8 bit value in A-register
00E6 A7    44               STA    TEMP,U       ...with start and stop bits
00E8 C6    08               LDB    #8
00EA 8D    C8               BSR    WBIZ         Generate start bit
00EC A6    44        WBY1   LDA    TEMP,U       Get value and isolate LSB...
00EE 84    01               ANDA   #1
00F0 26    04               BNE    WBY2         ...and output appropriate wave form
00F2 8D    C0               BSR    WBIZ         "0" data bit
00F4 20    02               BRA    WBY3

00F6 8D    C4        WBY2   BSR    WBIW         "1" data bit
00F8 A6    44        WBY3   LDA    TEMP,U       Adjust data value
00FA 47                     ASRA
00FB A7    44               STA    TEMP,U
00FD 5A                     DECB                Continue until 8 bits are done
00FE 26    EC               BNE    WBY1
```

```
0100 8D 0A          BSR    WBIT     Generate "stop" bit
0102 35 04          PULS   B        restore B-register
0104 39             RTS             Return
                *
                * Write Non-ASCII Bytes Subroutine
                *
0105 EB 84      XCBT ADDB  0,X      Update checksum
0107 A6 84          LDA    0,X      Get data value pointed to by X-register
0109 8D 03          BSR    WHLB
010B 30 01          LEAX   1,X      Update data pointer
010D 39             RTS             Return

010E 34 02      WHLB PSHS  A
0110 8D 04          BSR    WHL
0112 35 02          PULS   A
0114 20 04          BRA    WHP

0116 44         WHL  LSRA           Get high order nibble...
0117 44             LSRA            ...to low order nibble
0118 44             LSRA
0119 44             LSRA

011A 84 0F      WHP  ANDA  #$F      Mask out high order nibble
011C 8B 90          ADDA   #$90
011E 19             DAA             Convert to ASCII
011F 89 40          ADCA   #$40
0121 19             DAA

0122 20 C8          BRA    WBYT     Output value
                *
                * STRINGS
                *
0124 0A 0D 2A 2A MSG1 FCC  LF,CR,"** Kansas City (300b.) Writer for TC-3 **",LF,CR
0128 20 4B 61 6E
012C 73 61 73 20
0130 43 69 74 79
0134 20 28 33 30
0138 30 62 2E 29
013C 20 57 72 69
0140 74 65 72 20
0144 66 6F 72 20
0148 54 43 2D 33
014C 20 2A 2A 0A
0150 0D

0151 53 54 41 52      FCC  "START RECORDER, then input range to be written: ",EOT
0155 54 20 52 45
0159 43 4F 52 44
015D 45 52 2C 20
0161 74 68 65 6E
0165 20 69 6E 70
0169 75 74 20 72
016D 61 6E 67 65
0171 20 74 6F 20
0175 62 65 20 77
0179 72 69 74 74
017D 65 6E 3A 20
0181 04

0182 0A 0A 0D 4F MSG2 FCC  LF,LF,CR,"OK, Dump in progress...",EOT
0186 4B 2C 20 44
018A 75 6D 70 20
018E 69 6E 20 70
0192 72 6F 67 72
0196 65 73 73 2E
019A 2E 2E 04

019D 07 00 00 04 BELL FCC  BEL,0,0,EOT

                      END  BEGIN

0 ERROR(S) DETECTED

                * STRINGS
                *
00F5 0A 0A 0D 2A MSG1 FCC  LF,LF,CR,"** KC (300b) LOADER for TC-3 **",LF,CR
00F9 2A 20 4B 43
00FD 20 28 33 30
0101 30 62 29 20
0105 4C 4F 41 44
0109 45 52 20 66
010D 6F 72 20 54
0111 43 2D 33 20
0115 2A 2A 0A 0D
0119 57 68 65 6E      FCC  "When ready, hit "RETURN" key and hit "PLAY" on tape deck..."

011D 20 72 65 61
0121 64 79 2C 20
0125 68 69 74 20
0129 22 52 45 54
012D 55 52 4E 22
0131 20 6B 65 79
0135 20 61 6E 64
0139 20 68 69 74
013D 20 22 50 4C
0141 41 59 22 20
0145 6F 6E 20 74
0149 61 70 65 20
014D 64 65 63 6B
0151 2E 2E 2E
0154 04               FCB  EOT

0155 0A 0D 4F 4B MSG2 FCC  LF,CR,"OK, Load in progress...",EOT
0159 2C 20 4C 6F
015D 61 64 20 69
0161 6E 20 70 72
0165 6F 67 72 65
0169 73 73 2E 2E
016D 2E 04

016F 07 00 00 04 BELL FCC  BEL,0,0,EOT

                      END  BEGIN

0 ERROR(S) DETECTED
```

I've heard of the TC-3 being used at up to 9600 baud, but I found optimum reliability at 2400 baud. This allows Mike to load TSC 6809 BASIC (10K+) in under two minutes, compared to nearly 20 minutes at 300 baud.

ADVENTURE

MARK DATA

A REVIEW

by

PAUL E. PHELPS
111 DIVISION ST. 19
KING CITY, CALIF 93930

I admit to being pleased with opportunity to review these two new additions to the rather limited 6800 'Adventure' genre. When I called Mark Data Products to order the first of the games, "Calixto Island" and discovered I was receiving it and the second in the series, I could only wait with bated breath for the disk to arrive. Arrive it did and after several fun filled and challenging hours, I was able to solve the puzzels. Compared to the original these are not as complicated, but well worth the cost and no small challenge in themselves.

"Calixto Island" involves searching a building and series of locations for a treasure, which then must be brought back to the starting location to win. Not as simple as it sounds! The dialog seems well written and, occasionally even funny. The game movest smoothly and, because these are written in machine language, they are, to all intents and purposes, instant in response. I will give no clues to their solution. Suffice it to say that, if you purchase the disk from Mark Data Products, you won't be dissapointed in the challenge.

Personally, I liked the second adventure, "Black Sanctum" better than the first. I found it more challenging and it took longer to resolve and that is after having played "Calixto Island" first and, therefore, being a bit familiar with the tactics used in designing the puzzels. Perhaps being a Chaplain has a bearing on it, but the use of Midieval imagry and 'black magic' intrigued me and I thuroughly enjoyed it. My wife and I spent a pleasent afternoon

wandering around in search of the 'evilforce' and learning how to destroy it. Quite a challenge.

I look forward to future productions from Mark Data Products. In a phone conversation with the author, he indicated that there were other games in the works some for adulst, as these are, but also some for younger folk too.

There is, it seems to me, a need for some of the 'fun and games' aspects of hobby computers to be more available in our corner of the world. Mark Data Products has opened the door. I, for one, will look for many more hours of fun and challenge to walk through it (provided, of course, that I have found the key, or cleared the entrance, or ....).

These games are available for $16.95 a piece from Mark Data Products, 23082 Barquilla, Mission Viejo, California, 92691. At present they come in FLEX 2.0 disk format, but would, I suspect, run equally well in any other 6800 system (except for the game "save " function, of course). They are single entities and come with adaptation notes provided by the author. They run in 16k systems plus the DOS. If you decide to buy one or both, good luck on your travels! You will need it!

UNDERSTANDING SUBROUTINES PART 2 — ASSEMBLY LANGUAGE SUBROUTINES AND PARAMETERS

John F. Wakerly
Micro Systems Engineering
257 Castro Street, Suite 2E3
Mountain View, CA 94041

Part 1 of this three-part series discussed procedures and parameters in Pascal. Part 2 shows how these high-level concepts relate to assembly language subroutines and parameter-passing conventions using examples in Motorola 6809 assembly language.

John Wakerly is an independent consultant and a consulting associate professor at Stanford University. This three-part tutorial on subroutines in Pascal and 6809 assembly language is adapted from his recently-published book, Microcomputer Architecture and Programming, *copyright 1981, with permission of the publishers, John Wiley & Sons, Inc. (The book is also available directly from the author at MSE Books; see advertisement elsewhere in this issue.)*

JUGGLER

Recently we received a copy of the MICRO POWERS'S 'JUGGLER' game. It runs under TSC Extended BASIC© and works with the current SWTPC series of video terminals.

It is a game that requires a developing skill level and has been a hit with the office staff here. Fact is, they disabled the 'bell' code so that game playing was not quite so noticable.

For the user who needs an occasional relief from heavy programming and who might want to entertain the kids (small and large) this is a nice package.

JUGGLER may be ordered from: MICRO POWER Systems and Software, 1418 Thorndale, Chicago, IL 60660. Telephone 312 989-8585.

SUBROUTINE CALLING METHODS

In order to execute subroutines, a processor must have a means for a program to save a return address when the subroutine is called, and a way for the subroutine to jump to the return address when the subroutine is finished. Theoretically, subroutine return addresses could be handled by ordinary instructions, as shown in Table 1 for the 6809. Here the programmer has set up a convention for subroutine calling programs to save a return address in two reserved bytes at the beginning of the subroutine. Note that this convention cannot be used if the subroutine is stored in read-only memory!

TABLE 1 How to call subroutines in the 6809 without using JSR and RTS. By convention, a subroutine return address is deposited in the first two bytes of the subroutine. The first executable instruction begins in the third byte of the subroutine.

```
MAIN  ...            Main program.
      LDX   #RET1    Load X with the return address
      STX   SUBR       and store it in subroutine.
      JMP   SUBR+2   Jump to subroutine.
RET1  ...            Return here when subroutine done.
      ...
      LDX   #RET2    Save return address again.
      STX   SUBR
      JMP   SUBR+2   Jump to subroutine.
RET2  ...            Return here when subroutine done.
      ...
*
SUBR  RMB   2        Reserve two bytes for return addr.
      STA   P1       First executable instruction.
      ...
      LDX   SUBR     Get return address from loc. SUBR.
      JMP   ,X       Jump to address contained in X.
```

Because subroutines are used so often, all modern processors have special built-in instructions for calling subroutines and returning from them. A pushdown stack is the most appropriate data structure for saving subroutine return addresses, because it can store more than one return address when subroutines are nested. The number of levels of nesting is limited only by the size of the stack.

Most processors have a dedicated register (SP) that points into a stack of subroutine return addresses. The subroutine calling instructions (JSR or CALL) push return addresses onto the stack, and subroutine return instructions (RTS or RET) pop return addresses off the stack.

In the 6809, the JSR and RTS instructions provide for subroutine calls and returns in conjunction with the stack pointer register SP. Any program that uses subroutines is required to reserve a small area of memory for a push-down stack for return addresses. At the beginning of such a program SP must be initialized to point at this area using the LDS #addr instruction. As shown in Figure 1, SP points to the top item in the stack, or just past the stack area if the stack is empty. SP is decremented once before storing each byte on the stack, and incremented once after popping each byte. A return address occupies two bytes.

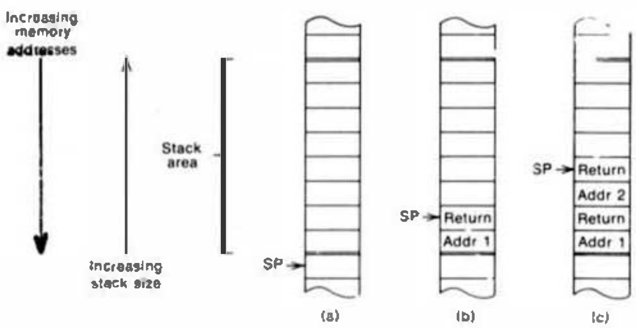


FIGURE 1 6809 return-address stack: (a) empty; (b) after one subroutine call; (c) after second (nested) subroutine call.

Table 2 outlines a Pascal program with two nested subroutines and the corresponding assembly language statements. The JSR addr instruction saves the address of the next instruction by pushing it onto the stack and then jumps to the instruction at location addr, the first instruction of the subroutine. At the end of the subroutine, RTS pops an address from the stack into PC, effecting a return to the original program sequence. Figure 2 shows the state of the stack after each instruction that affects it.

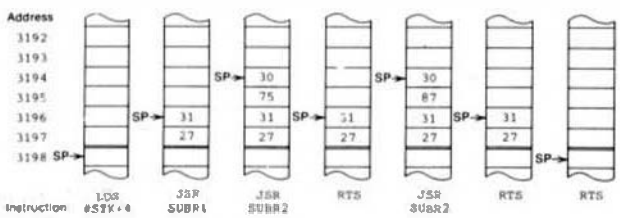


FIGURE 2 Stack contents after executing instructions in Table 2.

TABLE 2 A program with two nested subroutines. Both Pascal and corresponding 6809 assembly language statements are shown.

```
ADDR CONTENT LABEL OPCDE OPERAND COMMENTS

                   ORG   $3000   PROGRAM SubrExamples;
3000 ...     SUBR2 ...           PROCEDURE Proc2;
...  ...           ...             BEGIN
...  ...           ...               ...
3055 39            RTS             END;
*
3056 ...     SUBR1 ...           PROCEDURE Proc1;
...  ...           ...             BEGIN
...  ...           ...               ...
3072 BD 3000       JSR   SUBR2       Proc2; {Call Proc2}
3075 ...     RET2A ...               ...
...  ...           ...               ...
3084 BD 3000       JSR   SUBR2       Proc2; {Call Proc2
3087 ...     RET2B ...               ...          again}
...  ...           ...                 ...
3092 39            RTS               END;
*
3093 8F 3198 AIN   LDS   #STK+4  BEGIN {Main program}
...  ...           ...             ...
3124 BD 3056       JSR   SUBR1     Proc1; {Call Proc1}
3127 ...     RET1  ...             ...
...  ...           ...             ...
3191 7E 1000       JMP   $1000   END.
*
3194 (0004)  STK   RMB   4       {Reserve 4 bytes for
3198               END   AIN      two return addresses}
```

Processors that don't have a hardware stack pointer, such as the Texas Instruments 9900, have subroutine calling instructions that save the return address in a specified processor register. When subroutine calls are nested, it is up to the programmer to save the contents of this register in dedicated memory locations or in a stack.

SUBROUTINE PARAMETERS

In Part 1 we discussed several types of parameters used in Pascal procedures and functions. Table 3 shows the corresponding parameter types in assembly language subroutines.

TABLE 3 Parameters in Pascal and assembly language programs.

Pascal Parameter Type	Assembly Language Parameter Type
Value	Data value
Variable	Address of variable
Procedure or function name	Address of procedure or function

The parameter types specified in Pascal procedure and function definitions should be called *input parameters*, because they are passed from a calling program to a subroutine. However, subroutines can also pass results to a calling program. For example, a Pascal function returns one value to the calling program, while an assembly language subroutine can return many values. We'll call these values *output parameters*, *outputs*, or *results*.

To see the difference between variable parameters and value parameters, compare the swapping procedures in Table 4 and Table 5. In the first example, the main program passes the subroutine the addresses of the two variables; the subroutine accesses the variables by indirect addressing. In the second example, the main program passes copies of the variables to the subroutine, which the subroutine manipulates directly; the subroutine has no way to get at the original variables themselves.

TABLE 4 Swapping subroutine using variable parameters.

```
*       Swap two 8-bit variables x and y whose addresses
*       are passed in index registers X and Y.
SWAP    LDA   ,X        Put x and y into A and B.
        LDB   ,Y
        STA   ,Y        Save A and B into y and x.
        STB   ,X
        RTS
*       Main program -- swap the values of VARP and VARQ.
MAIN    ...
        LDX   #VARP     Load X with address of VARP.
        LDY   #VARQ     Load Y with address of VARQ.
        JSR   SWAP      Swap values of VARP and VARQ.
        ...
VARP    RMB   1         Reserve storage for VARP and VARQ.
VARQ    RMB   1
```

TABLE 5 Swapping subroutine using value parameters.

```
*         Swap two 8-bit values passed in regs A and B.
SWAP    STA    TEMP      Save value of A.
        TFR    B,A       Transfer B to A.
        LDB    TEMP      Put saved value of A into B.
        RTS
TEMP    RMB    1         Local variable.
*
*       Main program -- swap copies of VARP and VARQ.
MAIN    ...
        LDA    VARP      Load A with value of VARP.
        LDB    VARQ      Load B with value of VARQ.
        JSR    SWAP      Swaps copies of VARP and VARQ in
*                        A and B, originals are untouched.
        ...
VARP    RMB    1         Reserve memory for VARP and VARQ.
VARQ    RMB    1
```

The "Load Effective Address (LEA)" instruction found in many processors is very useful for passing variable parameters. This instruction loads an index register with the *address* of an operand, using a specified addressing mode. Suppose that a programmer wanted to swap VARP and SCORE[J], where SCORE[0..99] is an array of 100 bytes. Then the following main program statements in Table 4 would do the trick:

```
        LDX    #VARP     X now has address of VARP.
        LDY    J         Get index of array item J.
        LEAY   SCORE,Y   Y now has address of SCORE[J].
        JSR    SWAP
        ...
J       RMB    2         Storage for array index.
SCORE   RMB    100       Storage for 100-byte array.
```

Here the LEAY instruction loads Y with the address of SCORE[J] as computed at run time — the sum of J and the base address of the array SCORE.

From these examples, it is apparent that all parameters in assembly language programs are really "values." With "variable" parameters, the "value" that is passed just happens to be the address of a variable. How parameters are classified doesn't make much difference, as long as the subroutine and calling program agree on how the parameters will be used.

PASSING PARAMETERS IN REGISTERS AND MEMORY LOCATIONS

The simplest way for a program to pass parameters to a subroutine is to place them in the processor's registers. Likewise, the subroutine can return results to the calling program in the same way. This technique was used in the subroutines in the previous section. Of course, the programmer must ensure that the calling program and the subroutine agree on which register contains each parameter. The register allocation for parameters is usually stated in a comment at the beginning of the subroutine, as in the foregoing examples.

If a processor does not have enough registers to hold all of the input or output parameters of a subroutine, then dedicated memory locations may be used instead. These memory locations are associated with the subroutine itself, not the calling program, so that each calling program places inputs and retrieves outputs in the same pre-arranged locations. For example, the 6809 DIVIDE subroutine in Table 6 expects the caller to place input parameters in locations DVND and DVSR, and it places outputs in locations QUOT and REM.

PARAMETER AREAS

It is also possible to associate a parameter area with the calling program instead of with the subroutine. In this case, the calling program places parameters in the parameter area and passes the subroutine the base address of the parameter area. For example, Table 7 shows a new version of the DIVIDE subroutine in which the calling program uses register X to pass the base address of the parameter area. The subroutine may then use based addressing to access the parameters. In based addressing, an address register (e.g. X in the 6809) contains the base address of a data structure, while the instruction (e.g. LDB DVND+1,X) contains a constant offset (DVND+1 = 1) to a particular item in the data structure.

TABLE 6 6809 DIVIDE subroutine that passes parameters in dedicated memory locations.

```
*       SUBROUTINE DIVIDE
*       Divides a 16-bit unsigned number by an 8-bit
*       unsigned number, yielding 8-bit quotient and
*       remainder. The subroutine returns calls the
*       operating system on overflow and on attempts
*       to divide by zero.
*
DVND    RMB    2         Dividend.
DVSR    RMB    1         Divisor.
QUOT    RMB    1         Quotient.
REM     RMB    1         Remainder.
CNT     RMB    1         Local storage for counter.
*
DIVIDE  LDA    #8        Initialize count.
        STA    CNT
        LDA    DVND      Put dividend in A,B.
        LDB    DVND+1
        CMPA   DVSR      Will quotient fit in 1 byte?
        BLO    DIVLUP    Branch if it will.
        SWI              Else report overflow to opsys.
DIVLUP  ASLB             Left shift A,B with LSB:=0.
        ROLA             A carry here from MSB means
        BCS    QUOT1       high DVND definitely > DVSR.
        CMPA   DVSR      Compare high DVND with DVSR.
        BLO    QUOTOK    Quotient bit = 0 if lower.
QUOT1   INCB             Else set quotient bit to 1.
        SUBA   DVSR      And update high DVND.
QUOTOK  DEC    CNT       Decrement iteration count.
        BGT    DIVLUP    Continue until done.
        STA    REM       Store remainder.
        STB    QUOT      Store quotient.
        RTS              Return.
```

TABLE 7 6809 DIVIDE subroutine that uses a parameter area.

```
*       Input and output parameters are passed in a 5-byte
*       parameter area. The base address of the parameter
*       area is passed to the subroutine in X. Parameter
*       positions in the parameter area are defined below.
*
DVND    EQU    0         2-byte dividend.
DVSR    EQU    2         1-byte divisor.
QUOT    EQU    3         1-byte quotient.
REM     EQU    4         1-byte remainder.
*
DIVIDE  LDA    #8        Initialize count.
        STA    CNT
        LDA    DVND,X    Put dividend in A,B.
        LDB    DVND+1,X
        CMPA   DVSR,X    Will quotient fit in 1 byte?
        BLO    DIVLUP    Branch if it will.
        SWI              Else report overflow to opsys.
DIVLUP  ASLB             Left shift A,B with LSB:=0.
        ROLA             A carry here from MSB means
        BCS    QUOT1       high DVND definitely > DVSR.
        CMPA   DVSR,X    Compare high DVND with DVSR.
        BLO    QUOTOK    Quotient bit = 0 if lower.
QUOT1   INCB             Else set quotient bit to 1.
        SUBA   DVSR,X    And update high DVND.
QUOTOK  DEC    CNT       Decrement iteration count.
        BGT    DIVLUP    Continue until done.
        STA    REM,X     Store remainder.
        STB    QUOT,X    Store quotient.
        RTS              Return.
CNT     RMB    1         Local storage for counter.
```

TABLE 8 Program that calls DIVIDE.

```
*       Compute PDIVQ := P DIV Q; PMODQ := P MOD Q;
*         where all are 1-byte variables in memory.
DIVPQ   LDX   #PARMS      X points to parameter area.
        CLR   DVND,X      Clear high-order dividend.
        LDA   P           Store low-order dividend.
        STA   DVND+1,X
        LDA   Q           Store divisor.
        STA   DVSR,X
        JSR   DIVIDE      Do the division.
        LDA   QUOT,X      Save the quotient.
        STA   PDIVQ
        LDA   REM,X       Save the remainder.
        STA   PMODQ
        ...
P       RMB   1           Storage for P, Q, PDIVQ, PMODQ
Q       RMB   1             (all 1-byte variables).
PDIVQ   RMB   1
PMODQ   RMB   1
        ...
PARMS   RMB   5           Storage for parameter area.
```

Table 8 shows a program that calls this DIVIDE subroutine. Notice how the parameter area is associated with the calling program, not the DIVIDE subroutine. The calling program may use the same parameter area for other subroutines. If a main program calls many different subroutines, using a single parameter area saves memory compared with the alternative of allocating separate parameter variables for each subroutine. Of course, the parameter area must be big enough to hold the largest number of parameters used in any one subroutine.

Parameter areas are sometimes useful for subroutines that are stored in read-only memory (ROM). Since parameter values can't be stored into a ROM, it is more convenient to allocate storage for them with their calling programs in read/write memory (RWM).

One special form of parameter area is called an *in-line parameter area*. Here the parameters are stored in the calling program immediately following the subroutine calling instruction (JSR). The "return address" stored by JSR is really the address of the first parameter. Before returning, the subroutine must bump this return address value past the parameter area; presumably, the subroutine knows exactly how many memory bytes to skip over. In-line parameter areas should only be used if the actual parameters for any given subroutine call are always constant. If the actual parameters are variables, then the in-line parameter area must be modified each time the subroutine is called, impossible in ROM and undesirable even in RWM (some memory management systems enforce write protection on program areas).

STATIC AND DYNAMIC ALLOCATION

Storage allocation methods for subroutine parameters can be classified as static and dynamic. With *static allocation*, memory locations are reserved for the parameters of a particular subroutine or caller, and are unused at other times. Both versions of the DIVIDE subroutine above use static allocation. With *dynamic allocation*, parameters are stored in the designated area during subroutine execution, but the storage is available for other uses the rest of the time. Passing parameters in registers as in Tables 4 and 5 is the simplest form of dynamic allocation.

STACK-ORIENTED PARAMETER-PASSING CONVENTIONS

Placing parameters in a pushdown stack is a form of dynamic allocation used both by assembly language programmers and by compilers for several high-level languages, including Pascal and Forth. Parameters can generally use the same pushdown stack as return addresses, since most processors have instructions to push, pop, and access arbitrary data in the return-address stack.

Figure 3 shows a typical use of a return-address stack for passing parameters. The calling program reserves space on the stack for any output parameters and then pushes input parameters onto the stack. After the subroutine is called, the stack has the state shown in Figure 3(a). An address register FP is now used as a *stack frame pointer* (or *frame pointer*). The subroutine saves the old value of FP by pushing it onto the stack and then copies the value of SP into FP. The frame pointer provides a fixed reference for accessing parameters that does not change with SP as more items are pushed onto the stack. The region of the stack accessed during a subroutine's execution is called a *stack frame*.

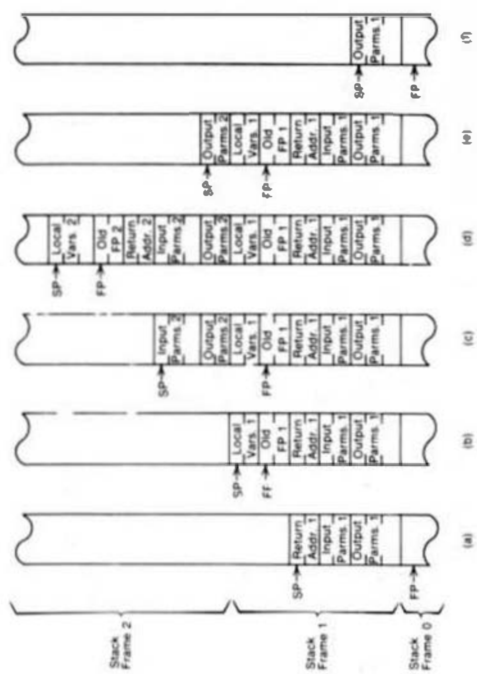


FIGURE 3 A pushdown stack with return address, parameters, and local variables: (a) just after calling SUBR1; (b) during execution of SUBR1; (c) just before a call to SUBR2; (d) during execution of SUBR2; (e) just after return from SUBR2; (f) just after return from SUBR1.

As shown in Figure 3(b), local variables can also be pushed onto the stack during a subroutine's execution and accessed by offsets from FP. If a second (nested) subroutine is called, parameters are again pushed onto the stack and a new stack frame is created, as shown in Figure 3(c,d). As each subroutine returns, it "cleans up" the stack by

(1) Removing its local data by setting SP equal to FP.

(2) Restoring the old value of FP by popping it from the stack.

(3) Removing the input parameters from the stack and jumping t the return address, leaving only the output parameters on the stack.

TABLE 9 DIVIDE routine that passes parameters on a stack.

```
*       The offsets below define positions of parameters
*       and local variables in the stack relative to a
*       frame pointer (register X).
CNT     EQU   -1
OLDFP   EQU   0          Old value of frame pointer.
RETADR  EQU   2          Return address.
DVSR    EQU   4          1-byte divisor (input).
DVND    EQU   5          2-byte dividend (input).
REM     EQU   7          1-byte remainder (output).
QUOT    EQU   8          1-byte quotient (output).
*
DIVIDE  PSHS  X          Push old frame pntr onto stack.
        TFR   S,X        Copy SP to X for new frame pntr.
        LDA   #8         Push initial count.
        PSHS  A
        LDA   DVND,X     Put dividend in A,B.
        LDB   DVND+1,X
        CMPA  DVSR,X     Will quotient fit in 1 byte?
        BLO   DIVLUP     Branch if it will.
        SWI              Else report overflow to opsys.
DIVLUP  ASLB             Left shift A,B with LSB:=0.
        ROLA             A carry here from MSB means
        BCS   QUOT1        high DVND definitely > DVSR.
        CMPA  DVSR,X     Compare high DVND with DVSR.
        BLO   QUOTOK     Quotient bit = 0 if lower.
QUOT1   INCB             Else set quotient bit to 1.
        SUBA  DVSR,X     And update high DVND.
QUOTOK  DEC   CNT,X      Decrement iteration count.
        BGT   DIVLUP     Continue until done.
        STA   REM,X      Store remainder.
        STB   QUOT,X     Store quotient.
        TFR   X,S        Remove local variables.
        PULS  X          Restore frame pointer.
        PULS  Y          Get return address, save in Y.
        LEAS  3,S        Remove input parms (SP := SP+3).
        JMP   ,Y         Return to addr contained in Y.
```

TABLE 10 Program that calls stack-oriented DIVIDE.

```
*       Compute PDIVQ := P DIV Q; PMODQ := P MOD Q;
*           where all are 1-byte variables in memory.
DIVPQ   LEAS  #-2,S      Make room for output parms.
        LDA   P          Push low-order dividend.
        PSHS  A
        CLR   A          Push high-order dividend = 0.
        PSHS  A
        LDA   Q          Push divisor.
        PSHS  A
        JSR   DIVIDE     Do the division.
        PULS  A          Pop remainder and store.
        STA   PMODQ
        PULS  A          Pop quotient and store.
        STA   PDIVQ
        ...
P       RMB   1          Storage for P, Q, PDIVQ, PMODQ
Q       RMB   1             (all 1-byte variables).
PDIVQ   RMB   1
PMODQ   RMB   1
```

The stack-oriented subroutine calling convention is illustrated by the DIVIDE subroutine in Table 9. Register X is used as the frame pointer. A program that calls DIVIDE is shown in Table 10. The state of the stack before the DIVIDE subroutine is called is shown in Figure 4(a). The calling program reserves two bytes on the stack for REM and QUOT and then pushes DVND and DVSR (Figure 4(b)) and calls DIVIDE. Then DIVIDE pushes the old value of the frame pointer and CNT onto the stack as shown in Figure 4(c). Upon return, the stack is cleaned up, leaving only the output parameters REM and QUOT as shown in Figure 4(d).

You may have noticed that the use of X as a frame pointer in Table 9 is somewhat superfluous since SP doesn't change throughout the subroutine's execution; we could have used SP instead of X by adding 1 to all the offsets. However, in a more general subroutine, intermediate results of expression evaluations and other computations might be temporarily pushed onto and popped from the stack, so that parameter offsets from SP would be continually changing. In this case, the fixed frame pointer (X) is much easier to use than SP.

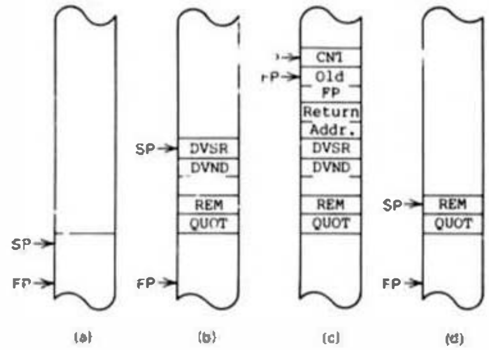


FIGURE 4 Stack contents during DIVPQ program: (a) at start; (b) just before calling DIVIDE; (c) after first four instructions of DIVIDE; (d) on return.

The DIVIDE subroutines in Tables 7 and 9 both access parameters in a "parameter area," using based addressing with offsets from X. The major difference is that the parameter area for Table 7 is allocated statically when the program is assembled, while the parameter area for Table 9 is created dynamically on the stack each time the subroutine is called. With static parameter areas we need one for each subroutine; with a stack we need only enough storage for the maximum number of parameters that are "active" when subroutines are nested. In a program with many subroutines the stack convention could yield significant memory savings.

Procedures and functions in Pascal pass parameters using a stack-oriented convention similar to the one described above. Local variables are also stored on the stack. This explains why the values of local variables are not preserved between successive invocations of the same procedure—the stack pointer may start at a different position on each invocation, so that the variables could actually be stored in different memory locations on different invocations.

ANOTHER EXAMPLE: QUEUE SUBROUTINES

To conclude this part, we give a set of 6809 subroutines that manipulate queues. These subroutines use a parameter area called a "queue descriptor table," and their documentation includes "prologues" that explain how the subroutines work. In keeping with the philosophy that programs should be self-documenting, we leave you to read Table 11.

TABLE 11 Queue manipulation subroutines for the 6809.

```
* QUEUE MODULE
*
* This module has three subroutines for manipulating
* queues of 8-bit bytes. A queue is defined by a queue
* descriptor table and a storage block as shown below.
*
*                                        QUEUE STORAGE BLOCK
*          ---------------------         --------------
* QDTBL   |    QHEAD  (word)     |  ------> |  (byte)    |
*         |----------------------|  |      |------------|
*         |    QTAIL  (word)     |  |      |            |
*         |----------------------|--       |    o o o   |
*         |    QSTRT  (word)     |<--      |            |
*         |----------------------|         |------------|
*         |    QEND   (word)     |-------> |  (byte)    |
*          ---------------------         --------------
```

```
*
* Offsets in descriptor table:
*
QHEAD  EQU   0
QTAIL  EQU   2
QSTRT  EQU   4
QEND   EQU   6
*
* In this table, the first two words are constants,
* initialized at load time, that give the starting and
* ending addresses of the block of storage (buffer)
* reserved for the queue itself.  The third and fourth
* words are reserved to store the queue head and tail
* (which are absolute memory addresses), and are
* manipuleted by the subroutines.
*
* If a program defines several queues, it allocates a
* separate queue descriptor table and storage block for
* each one.  For example, the statements below define
* a 5-byte queue Q1 and a 100-byte queue Q2:
*
*Q1BLK RMB   5        Storage block for Q1.
*Q1END EQU   *-1      Last location in Q1 storage block.
*Q1DT  RMB   4        Q1 descriptor table: QHEAD, QTAIL.
*      FDB   Q1BLK,Q1END                   QSTRT, QEND.
*Q2BLK RMB   100      Storage block for Q2.
*Q2END EQU   *-1      Last location in Q2 storage block.
*Q2DT  RMB   4        Q2 descriptor table: QHEAD, QTAIL.
*      FDB   Q2BLK,Q2END                   QSTRT, QEND.
*
* Subroutines are provided to initialize a queue (QINIT),
* enqueue a byte (QENQ), and dequeue a byte (QDEQ).
* Each subroutine must be passed the address of the
* descriptor table for the queue to be manipulated.
*
* SUBROUTINE QINIT -- Initialize a queue to be empty.
*
* INPUTS
*     #QDTBL -- The address of the queue descriptor table
*               for the queue to be initialized,
*               passed in register X.
* OUTPUTS, GLOBAL DATA, LOCAL DATA -- None
* FUNCTIONS
*     (1) Initialize the queue to empty by setting QHEAD
*         and QTAIL in QDTBL equal to the first address
*         in the queue buffer.
* REGISTERS AFFECTED -- CC
* TYPICAL CALLING SEQUENCE
*       LDX   #Q1DT
*       JSR   QINIT
*
QINIT  PSHS  Y             Save Y so not affected.
       LDY   QSTRT,X       Put buffer starting addr into Y.
       STY   QHEAD,X       Store into QHEAD and QTAIL.
       STY   QTAIL,X
       PULS  Y             Restore Y.
       RTS                 Done, return.
*
* SUBROUTINE QENQ -- Enqueue one byte into a queue.
*
* INPUTS
*     #QDTBL -- The address of the queue descriptor table
*               for the queue to be initialized,
*               passed in register X.
*     QDATA  -- The byte to be enqueued, passed in reg A.
* OUTPUTS
*     QFULL  -- 1 if the queue is already full, else 0;
*               passed in condition bit Z.
* GLOBAL DATA, LOCAL DATA -- None.
* FUNCTIONS
*     (1) If the queue described by QDTBL is full,
*         set QFULL to 1.
*     (2) If the queue described by QDTBL is not full,
*         enqueue QDATA and set QFULL to 0.
* REGISTERS AFFECTED -- CC
* TYPICAL CALLING SEQUENCE
*       LDX   #Q1DT         Enqueue byte ABYTE.
*       LDA   ABYTE
*       JSR   QENQ
*       BEQ   OVFL          Branch if queue is full.
*
QENQ   PSHS  Y             Save Y so not affected.
       LDY   QTAIL,X       Get queue tail.
       LEAY  1,Y           Bump to next free location.
       CMPY  QEND,X        Wrap-around?
       BLS   QENQ1
       LDY   QSTRT,X       Reinitialize on wrap-around.
QENQ1  CMPY  QHEAD,X       Queue already full?
       BEQ   QENQ2         Return with Z=1 if full.
       STA   [QTAIL,X]     Else store QDATA at old tail.
       STY   QTAIL,X       Update tail.
       ANDCC #$FB          Set Z:=0 since not full.
QENQ2  PULS  Y             Restore Y.
       RTS                 Return.
* SUBROUTINE QDEQ -- Dequeue one byte from a queue.
*
* INPUTS
*     #QDTBL -- The address of the queue descriptor table
*               for the queue to be manipulated,
*               passed in register X.
* OUTPUTS
*     QEMPTY -- 1 if the queue is empty, else 0;
*               passed in condition bit Z.
*     QDATA  -- The byte dequeued, passed in register A.
* GLOBAL DATA, LOCAL DATA -- None.
* FUNCTIONS
*     (1) If the queue described by QDTBL is empty,
*         set QEMPTY to 1.
*     (2) If the queue described by QDTBL is not empty,
*         daqueue QDATA and set QEMPTY to 0.
* REGISTERS AFFECTED -- A, CC
* TYPICAL CALLING SEQUENCE
*       LDX   #Q1DT         Dequeue a byte into ABYTE.
*       JSR   QDEQ
*       BEQ   UNDFL         Branch if queue is empty.
*       STA   ABYTE
*
QDEQ   PSHS  Y             Save Y so not affected.
       LDY   QHEAD,X       Get queue head.
       CMPY  QTAIL,X       Queue empty?
       BEQ   QDEQ2         Return with Z=1 if empty.
       LDA   ,Y+           Read QDATA byte from queue and
*                            bump Y to next item in queue.
       CMPY  QEND,X        Wrap-around?
       BLS   QDEQ1
       LDY   QSTRT,X       Reinitialize on wrap-around.
QDEQ1  STY   QHEAD,X       Save new value of head.
       ANDCC #$FB          Set Z:=0 since not empty.
QDEQ2  PULS  Y             Restore Y.
       RTS                 Return.
```

REFERENCES

A fascinating history of subroutines and related concepts can be found in Knuth's *Fundamental Algorithms* [Addison-Wesley, 1973 (second edition), pp. 225–227]. Instructions for calling subroutines were included in all of the early digital computers, although it was not until the 1960s that a pushdown stack was used to store return addresses (in the B 5000 [Lonergan and King, "Design of the B 5000 system," *Datamation*, Vol. 5, No. 7, May 1961, pp. 28–32; also in Bell and Newell's *Computer Structures*, McGraw-Hill, 1971]).

The architecture of the B 5000 supported parameter passing on a stack; Algol and other related high-level languages have popularized the use of the stack. More recently, special instructions have been provided in new computer architectures to facilitate parameter passing on a stack (RET n in the 8086 and LINK and UNLK in the 68000).

The correspondence between parameter-passing conventions and run-time storage allocation and operations in high-level languages is explained in *Compiler Construction for Digital Computers* by David Gries [Wiley, 1971].

In the third and final part of this series, we'll introduce the advanced concepts of recursion and coroutines, giving examples in both Pascal and assembly language.

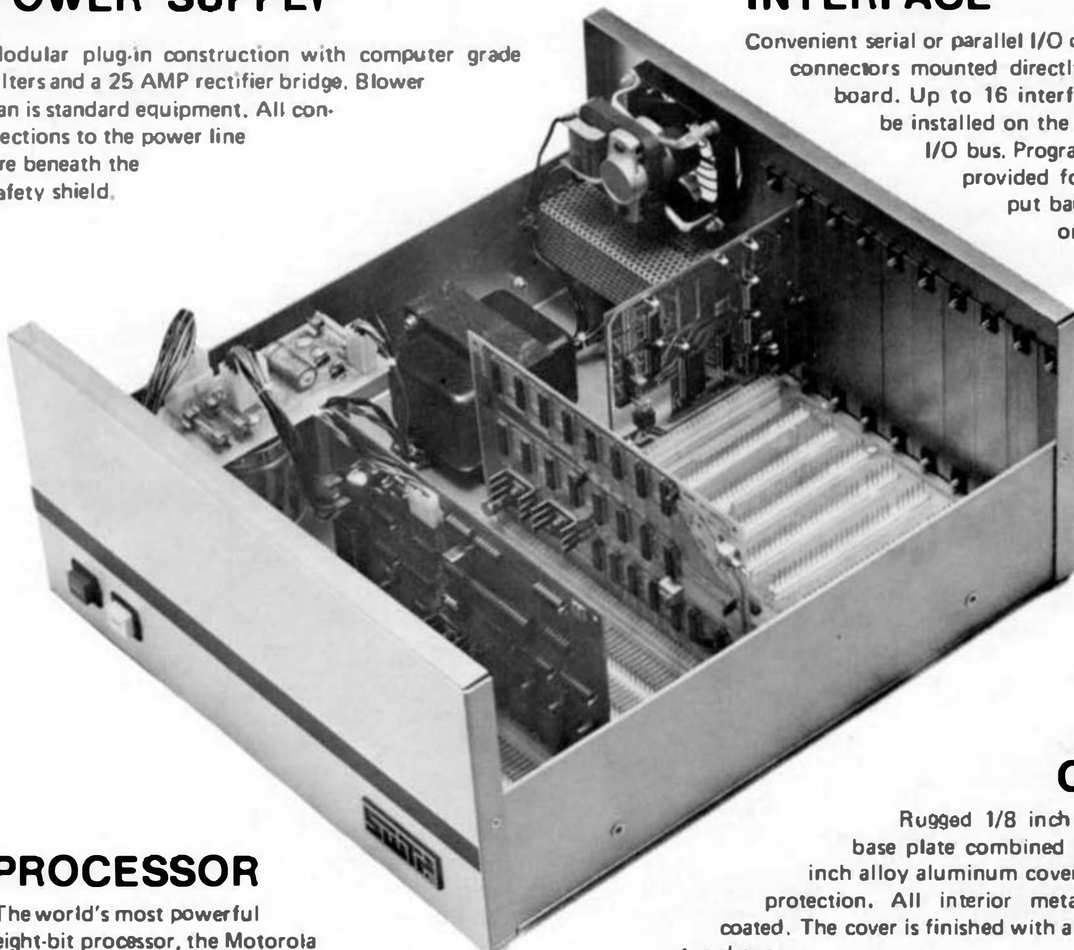
WE HAVE A 6809 FOR YOU

POWER SUPPLY

Modular plug-in construction with computer grade filters and a 25 AMP rectifier bridge. Blower fan is standard equipment. All connections to the power line are beneath the safety shield.

INTERFACE

Convenient serial or parallel I/O cards have DB-25 connectors mounted directly on the circuit board. Up to 16 interface devices may be installed on the address decoded I/O bus. Programming strips are provided for input and output baud rate selection on each port. All outputs are fully buffered.



CABINET

Rugged 1/8 inch alloy aluminum base plate combined with a solid 1/8 inch alloy aluminum cover for unsurpassed protection. All interior metal is conversion coated. The cover is finished with a super tough textured epoxy.

PROCESSOR

The world's most powerful eight-bit processor, the Motorola MC6809, plus 2K byte monitor ROM that is 2716 EPROM compatible and full buffering on all output lines. Built-in multiuser capability, just add I/O cards to operate a multi-terminal system.

MEMORY— You can purchase the computer with either 8K bytes of RAM memory (expandable to 56K), or with the "S" series 64K bytes of RAM memory expandable to 768 K.

PERIPHERALS—The wide range of peripheral hardware that is supported by the 6809 includes: dot matrix printers (both 80 and 132 column), IBM Electronic 50 typewriter, daisy wheel printers, 5-inch floppy disk system, 8-inch floppy disk systems and a 16 megabyte hard disk.

SOFTWARE— The amount of software support available for the 6809 is incredible when you consider that it was first introduced in June, 1979. In addition to the FLEX9 operating system, we have a Text Editor, Mnemonic Assembler, Debug, Sort-Merge, BASIC, Extended BASIC, MultiUser BASIC, FORTRAN, PASCAL and PILOT.

69/K Computer Kit with 8K bytes of memory .... $ 575.00
69/A Assembled Computer with 8K bytes of memory .... $ 695.00
09/ Assembled Computer "S" series with 64K bytes of memory $1,595.00

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

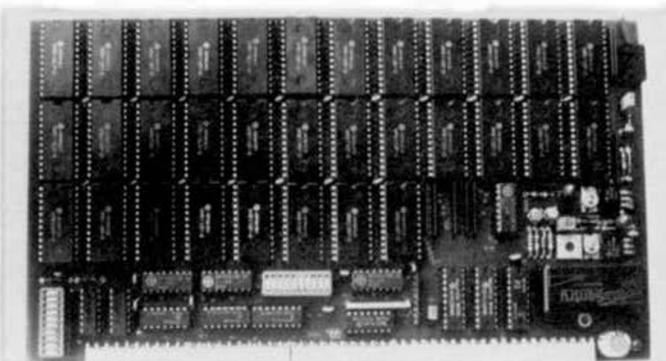
BIT Bucket

DON, JUST WANT TO ADVISE YOU THAT I RECEIVED A NEW VERSION OF 'ADVENTURE' TODAY THAT WORKS WITH FLEX 9. (6809 FLX 8" DISK). AS I MENTIONED IN MY PEVIOUS CARD, THE EARLIER VERSON TRIED TO LOAD AT $8300 ABOVE $7FFF. THE FIX HE USES INCLUDES ADVENTURE. SYS WHICH HAS A COMPANION LOADER PROGRAM WHICH IS HARDWARE INDEPENDENT REQUIRING ONLY MEMORY FROM 0-$7FFF AND $C000 TO $DFFF (6809). IT IS WORKING FINE AND I AM SATISFIED THAT THEIR INTENT IS TO HAVE SATISFIED CUSTOMERS! HE ALSO OFFERED PERSONAL ASSISTANCE BY PHONE OR LETTER IN CASE OF FURTHER DIFFICULTY. THANKS FOR YOUR INTEREST AND THE SERVICE YOU AND OUR MAGAZINE PROVIDES!
JOHN NIERSTE, 4205 W FARRAND, CLIO, MI 48420.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

Press Release

GIMIX INC. announces their new 64K BYTE CMOS STATIC RAM BOARD with battery back-up for the SS50/C bus. This board is fully compatible with any of the 6800/6809 DMA techniques. They are guaranteed for 2MHz operation with no wait states or clock stretching required. Ultra-low power CMOS RAM requires less than ¼ AMP (250 Ma.) at 8V for a fully populated 64K BYTE board. It is non-volatile using an on-board nickel-cadmium battery. The board retains data even with system power removed. With the battery fully charged, the contents of the memory remains intact for a minimum of 21 days.



The GIMIX 64K BYTE STATIC RAM BOARD is ideally suited to a wide variety of applications. The battery back-up feature is useful where data loss due to power failure cannot be tolerated, or as a replacement for disk or tape storage where conditions such as environment prohibit their use. Since the entire board can be hardware write protected by a switch located at the top of the board, it can also be used to emulate PROM or ROM memory. This is especially useful during firmware development where frequent software changes must be made.

When the board is used in conjunction with a device such as the GIMIX MISSING CYCLE DETECTOR BOARD, critical data can be stored and system integrity maintained during either expected or unexpected power outages.

The GIMIX 64K BYTE STATIC MEMORY is available from stock and is priced at $1088.64 and a 56K version, socketed for 64K is $994.56

luciddata

SOFTWARE LICENCE POLICY ANNOUNCEMENT

LUCIDATA has recently made two important changes to its software licence policy. The changes have been introduced, in consideration of the requirements of our users, to encourage the use of Pascal for writing application packages and implementing stand alone systems, and in recognition of the impracticability of enforcing compliance with an inflexible licence policy.

I Registered owners who have developed application software using LUCIDATA Pascal and wish to supply it to third parties, may include the LUCIDATA run-time system, with their application software, provided that the following conditions are met:

i) The LUCIDATA copyright line, printed at run-time initialisation, is not suppressed

ii) The LUCIDATA run-time system is only supplied bound with the application program as a command file, i.e. it is not usable by the third party for any other purpose or program. (The RESTART utility may be included as is).

II The price of the LUCIDATA ROM package has been increased to include an OEM licence fee. Registered owners of this product are free to include the run-time modules provided, in the firmware of their application systems, for sale to third parties.

Other than as stated above, resale of LUCIDATA software products remain governed by dealership or third party licence agreements. Unauthorised copying and resale of copyright software products is considered to be simple theft.

LUCIDATA LTD.
P.O. Box 128
CAMBRIDGE
CB2 5EZ
ENGLAND

May 1981

Jeff Luton
ModemPlus, Inc.
217 Trinity Ave.
Decatur, GA. 30030

3/30/81

A $50 TMS2516/TMS2532 EPROM PROGRAMMER!

Now that the era of tri-voltage EPROMS (Erasable Programmable Read Only Memories) has faded into history and the single voltage EPROMS are rapidly increasing in capacity and decreasing in price, the need arises for a convenient and economical means of programming the new devices on standard 6800 bus systems. This article and assembly code listing provide a methodology for programming - reading - copying - verifying Texas Instrument 2516 and 2532 compatible EPROMS using a modified SWTPC MP-R board (A-10 bus slot 7) and an additional toggle switch.

As can be seen from the updated MP-R schematic, the circuit modifications are straightforwardly simple and consist of the following steps:

1. change electrolytic capacitor C2 from 470 ufd. to 1 ufd.
2. mount S.P.D.T toggle switch & label 2516-2532 switch positions (the upper left corner of the MP-R board, previously occupied by the large C2 cap, is a good location for switch mounting)
3. cut land between IC3 pin 17 and resistor R7
4. cut land between IC3 pin 8 and IC4 pin 10
5. patch-wire IC3 pin 8 to open-end of resistor R7
6. patch-wire IC4 pin 10 to common-middle switch leg
7. patch-wire IC3 pin 17 to 2532 switch position
8. patch-wire IC3 pin 5 to 2516 switch position

Operation of the program listing code provides the most needed EPROM utility functions, including:

1. Program TMS2516 EPROM from RAM source data
2. Program TMS2532 EPROM from RAM source data
3. Verify TMS2516/TMS2532 data against RAM data
4. Read TMS2516/TMS2532 data into RAM source

Each of the utility functions operate on system RAM data at locations ranging from $0000-$07FF (2K) for the TMS2516 and RAM locations $0000-$0FFF (4K) for the TMS2532. It is not necessary to perform operations on the entire 2K or 4K RAM range; the selected EPROM may be partially read or written by specifying inclusive starting and ending addresses in response to the program operating messages. For example, a single EPROM location may be programmed by entering identical start-end addresses within the range of the selected EPROM (2K or 4K).

The 50 msec. EPROM programming pulse timing is controlled by the DELAY routine's initial count value; a $2900 value is used for a default 1.5 Mhz. 6800 system. Counter values of $1760 and $34E0 may be used for 1 Mhz. and 2 Mhz. systems, respectively.

MC6800 mnemonics and registers are used throughout the assembly code listing to enable re-assembly on any of the 68XX family machines; use of the extra 6809 registers reduces the size and complexity of the code but make conversion to 6800 more difficult.

The utility program's code generation was performed on a Smoke Signal Chieftain 6809 machine with frequent use of inherent system monitor [illegible] routines for the FCB statements at the beginning of the listing. All of the monitor routines (except maybe INCHK which can be easily replaced) have counterparts with similar names in the other popular monitors: Mikbug-Swtbug-Sbug...etc.

Scratchpad RAM allocation at $A000 is easily re-assembled for systems with RAM at $A000 or $C000. Similarly, the actual EPROM utility program itself may be re-assembled to reside at any convenient address - I have blown it into EPROM at $F000 in my system. Reference Peter Stark's article in the 3/81 Kilobaud magazine for information on modifying the MP-R to handle Intel 2732 compatible EPROMs.

```
        ***** C D S   U T I L I T Y   R O U T I N E S   PROGRAMMED BY J.A.CUTON  3/80  ****
                   OPT    [illegible]

         F8AB     A BADDR  EQU    $F8AB     MON47D GET 4 KYBD CHARS & CONVT TO 4 HEX CHARS
         F850     A PDATA  EQU    $F850     MON47D DISPLAY STRING VIA XREG W/04 TERMINATOR
         F833     A INEEE  EQU    $F833     MON47D ROUTINE TO GET SINGLE KYBD I/P CHAR
         F01C     A INCHK  EQU    $F01C     MON47D ROUTINE CHKS FOR KYBD I/P CHAR
         F8B3     A OUT2HS EQU    $F8B3     MON47D DISPLAY HEX BYTE VIA XREG
         F8B1     A OUT4HS EQU    $F8B1     MON47D DISPLAY 2 HEX BYTES ROUTINE
         F8[illegible]     A PCRLF  EQU    $F8[illegible]     MON47D PRINT CR/LF ON SCREEN ROUTINE

F000                       ORG    $F000     SCRATCH-PAD RAM

F000     0000     A STRADR FDB    $0000     STARTING ADDR OF MEM DATA
F002     0000     A ENDADR FDB    $0000     ENDING ADDR OF MEM DATA
F004     0000     A ACCTRL FDB    $0000     TEMP DATA STORAGE
F006     0000     A TEMPXR FDB    $0000     DISCREPANCY COUNTER - VERIFY
F008     00       A FRMCNT FCB    $00       2516-2532 SELECTION FLAG
F009     00       A TEMPAR FCB    $00       FIRST PASS FLAG - DELAY
F00A     00       A BYTECT FCB    $00       CHARS PER DISPLAY LINE COUNTER

B000                       ORG    $B000     [illegible]

                   ***************************************************************************
                   * [illegible] THE [illegible] 2516 OR 2532 I [illegible] EPROMS VIA MODIFIED [illegible] BOARD   *
                   * PROGRAMMED BY J.A.CUTON  11/24/80                                       *
                   ***************************************************************************
                   *                                                                         *
                   * VERIFICATION & READ EPROM ROUTINES MAY BE SUSPENDED-RESUMED VIA CTRL/C CMDS: *
                   * 1ST CTRL/C SUSPENDS CURRENT ROUTINE; 2ND CTRL/C RESUMES INTERRUPTED ROUTINE; *
                   * ENTRY OF ANY CHARACTER OTHER THAN 2ND CTRL/C WILL RETURN TO SELECTION MODE.  *
                   ***************************************************************************

         F7FC     A DRA    EQU    $F7FC     PIA IN SLOT 7 OF SWTPC I/O BUS
         F7FD     A CRA    EQU    DRA+1
         F7FE     A DRB    EQU    DRA+2
         F7FF     A CRB    EQU    DRA+3

                   ***************************************************************************
                   * PIA REGISTER/BIT CHART:                                                 *
                   ***************************************************************************
                   * 2516:                                                                   *
                   * DATA REG A:  BIT7   BIT6   BIT5   BIT4   BIT3   BIT2   BIT1   BIT0      *
                   * CB2 STROBE:  +25V   CE     PGM    +5V    A10    A9     A8     A8        *
                   * READ MODE:   0      0      0      0      USED                           *
                   * PROM MODE:   1      1      PULSED 1                                     *
                   * IDLE MODE:   0      0      0      0                                     *
                   * CA2 STROBE:  A7     A6     A5     A4     A3     A2     A1     A0        *
                   ***************************************************************************
                   * DATA REG B:  D7     D6     D5     D4     D3     D2     D1     D0        *
                   ***************************************************************************

                   ***************************************************************************
                   * PIA REGISTER/BIT CHART:                                                 *
                   ***************************************************************************
                   * 2532:                                                                   *
                   * DATA REG A:  BIT7   BIT6   BIT5   BIT4   BIT3   BIT2   BIT1   BIT0      *
                   * CB2 STROBE:  +25V   PGM    A11    +5V    A11    A10    A9     A8        *
                   * READ MODE:   0      0      USED   1                                     *
                   * PROM MODE:   1      PULSED 0      1                                     *
                   * IDLE MODE:   0      0      0      0                                     *
                   * CA2 STROBE:  A7     A6     A5     A4     A3     A2     A1     A0        *
                   ***************************************************************************
                   * DATA REG B:  D7     D6     D5     D4     D3     D2     D1     D0        *
                   ***************************************************************************
B000 7F  F7FD     A INSTRT CLR    CRA       INIT PIA - SELECT DIR REGS
B003 7F  F7FF     A        CLR    CRB
B006 86  FF       A        LDAA   #$FF
B008 B7  F7FC     A        STAA   DRA       SETUP ALL 16 O/P LINES
B00B B7  F7FE     A        STAA   DRB
B00E 86  34       A        LDAA   #$34
B010 B7  F7FD     A        STAA   CRA       SELECT DATA REGS & SETUP CA2/CB2 O/PS
B013 B7  F7FF     A        STAA   CRB
B016 4F                  IDLE   CLRA              SETUP & [illegible] CTRL BITS
B017 BD  B1BA     A        JSR    STROBE    GOSUB TO STROBE CTRL BYTE INTO DRA VIA CB2
B01A CE  B24F     A INSTRT LDX    #TITLE    POINT TO SELECTION INSTRUCTIONS
B01D BD  F850     A        JSR    PDATA     GOSUB TO DISPLAY MESSG ON CONSOLE
B020 CE  B233     A        LDX    #SELECT   POINT TO SELECTION PROMPT $
B023 BD  F850     A        JSR    PDATA     GOSUB TO DISPLAY IT FOR OPERATOR
B026 BD  F833     A        JSR    INEEE     GOSUB TO GET KYBD CHAR...
B029 7F  F00A     A        CLR    BYTECT    ZIP THE CHARS PER DISPLAY LINE COUNTER
B02C 81  35       A        CMPA   #'5       CHK OVER HI-LIMIT CMD?
B02E 24  EA  B01A          BCC    INSTRT    REPROMPT INVALID I/P IF SO...ELSE
B030 81  30       A        CMPA   #'0       CHK LESS THAN LO-LIMIT CMD?
B032 23  E6  B01A          BLS    INSTRT    REPROMPT IF SO...ELSE
B034 36  02              [illegible] PSHA           STORE INPUT CMD CHAR &
B036 CE  B30A     A        LDX    #PARAMS   POINT TO PARAMETER ENTRY MESSG
B039 BD  F850     A        JSR    PDATA     DISPLAY IT $
B03C BD  F8AB     A        JSR    BADDR     GOGET 4HEX CHARS FROM OPERATOR
B03F FF  F000     A        STX    STRADR    STORE START MEM ADDR CHARS
B042 CE  B36B     A        LDX    #ENDA     POINT TO END ADDR MESSAGE $
B045 BD  F850     A        JSR    PDATA     DISPLAY ENTRY PROMPT
B048 BD  F8AB     A        JSR    BADDR     GET END ADDR CHARS FROM OPERATOR
B04B 08                    INX              BUMP THE MEM END+1
B04C FF  F002     A        STX    ENDADR    STORE FOR END OF OPERATION CHK
B04F 32  00                PULA             RESTORE ORIG CMD CHAR $
B050 81  31       A        CMPA   #'1       CHK PROGRAM 2516 EPROM CMD?
B052 27  0E  B064          BEQ    PROGR2    GOTO DO IT IF SO...ELSE
B054 81  32       A        CMPA   #'2       CHK PROGRAM 2532 EPROM CMD?
B056 27  0E  B068          BEQ    PROGR4    GOTO DO IT IF SO...ELSE
B058 81  33       A        CMPA   #'3       CHK VERIFY EPROM CMD?
B05A 27  2C  B088          BEQ    VERIFY    GOTO DO IT IF SO...ELSE
B05C 81  [illegible]       A        CMPA   #'4       CHK READ EPROM CMD?
B05E 27  [illegible]  B0BB          BEQ    READOP    GOTO DO IT IF SO...ELSE
B060 20  B4  B016          BRA    IDLE      IF NOT, INVALID CMD TRY AGAIN...

                   ************  *************************************************************

B064 86  01       A PROGR2 LDAA   #1        SET 2516 SELECTION DESIGNATOR
B066 20  01  B069          BRA    STPROG    FOR 2K X 8 BIT EPROM

B068 4F                  PROGR4 CLRA              GET 2532 SELECTION DESIGNATOR
B069 B7  F008     A STPROG STAA   FRMCNT    & SET FLAG FOR SPECIFIED EPROM
B06C 7F  F009     A        CLR    TEMPAR    ZIP THE 1ST PASS THRU PROGRAMMING FLAG
B06F 7C  F009     A        INC    TEMPAR    THEN SET IT AS INDICATOR
B072 BD  B1CB     A        JSR    GETRDY    GOTO CHK FOR OPERATOR RDY...
B075 BD  B0D5     A CONTPG JSR    PROM      GOSUB TO PROGRAM-VERIFY 1 BYTE......
B078 BD  B1F3     A        JSR    [illegible]    GOSUB TO BUMP MEMPTR
B07B BC  F002     A        CPX    ENDADR    DONE W/ALL SPECIFIED MEMORY?
B07E 26  F5  B075          BNE    CONTPG    CONTINUE IF NOT...BUT
B080 CE  B3C1     A        LDX    #PROGCOM  OTHERWISE, POINT TO COMPLETED MESSG
B083 BD  F850     A        JSR    PDATA     GOSUB TO DISPLAY &
B086 20  8E  B016          BRA    IDLE      GOTO SET IDLE MODE...

                   ***************************************************************************
B088 BD  B1CB     A VERIFY JSR    GETRDY    GOSUB TO CHK OPERATOR RDY...
B08B CE  0000     A        LDX    #0        ZIP X REG & USE TO
B08E FF  F006     A        STX    TEMPXR    CLR DISCREPENCY COUNTER
B091 BD  B14F     A CONTVF JSR    VRFY      CHECK 1 BYTE OF RAM/EPROM...
B094 BD  B1F3     A        JSR    BMPCHR    UPDATE MEMPTR
B097 BC  F002     A        CPX    ENDADR    DONE W/ALL SPECIFIED DATA?
B09A 26  F5  B091          BNE    CONTVF    CONTINUE VERIFY IF NOT...BUT
B09C CE  B3D8     A        LDX    #VERCOM   OTHERWISE, POINT TO COMPLETE MESSG $
B09F BD  F850     A        JSR    PDATA     GOSUB TO DISPLAY
B0A2 7D  F006     A        TST    TEMPXR    ANY MIS-MATCHES DURING VERIFY?
B0A5 26  05  B0AC          BNE    NOMVER    GOTO DISPLAY NUMBER IF SO...ELSE
B0A7 7D  F007     A        TST    TEMPXR+1  CHK LO-ORDER COUNTER BYTE
B0AA 27  0C  [illegible]          BEQ    DTMVER    PROCEED TO RETURN IF NO DISCREPS...BUT
B0AC CE  B430     A NOMVER LDX    #NOMMSG   OTHERWISE, POINT TO MESSG &
B0AF BD  F850     A        JSR    PDATA     DISPLAY IT
B0B2 CE  F006     A        LDX    #TEMPXR   POINT TO MIS-MATCH COUNT &
B0B5 BD  F8B1     A        JSR    OUT4HS    DISPLAY IT 100
B0B8 7E  B016     A RTNVER JMP    IDLE      EXIT TO IDLE MODE...
                   ***************************************************************************

                   ***************************************************************************
B0BB BD  B1CB     A READOP JSR    GETRDY    GOSUB TO CHK OPERATOR RDY...
B0BE BD  B177     A CONTRD JSR    READ1     GOSUBS TO READ AN EPROM BYTE
B0C1 BD  B19F     A        JSR    READ2
B0C4 BD  B1F3     A        JSR    BMPCHR    UPDATE MEMPTR
B0C7 BC  F002     A        CPX    ENDADR    DONE W/ READ?
B0CA 26  F2  B0BE          BNE    CONTRD    CONTINUE IF NOT...BUT
B0CC CE  B3F6     A        LDX    #REDCOM   OTHERWISE, POINT TO COMPLETE MESSG
B0CF BD  F850     A        JSR    PDATA     GOSUB TO DISPLAY IT
B0D2 7E  B016     A        JMP    IDLE      THEN EXIT TO IDLE MODE...

                   ***************************************************************************

                   ***************************************************************************
                   * MAJOR SUBROUTINES:

B0D5 BD  B1DC     A PROM   JSR    DSPLAY    GOSUB TO DISPLAY ADDR NOW IF REQ'D
B0D8 B6  F7FF     A        LDAA   CRB
B0DB 84  FB       A        ANDA   #%11111011 SELECT DIR REG
B0DD B7  F7FF     A        STAA   CRB
B0E0 C6  FF       A        LDAB   #$FF
B0E2 F7  F7FE     A        STAB   DRB       SELECT ALL 8 O/P LINES FOR DATA
B0E5 8A  04       A        ORAA   #%00000100
B0E7 B7  F7FF     A        STAA   CRB       SELECT DATA REG
B0EA FE  F000     A        LDX    STRADR    GET CURRENT ADDR
B0ED A6  00       A        LDAA   0,X       GET DATA BYTE FROM RAM
B0EF B7  F7FE     A        STAA   DRB       SEND DATA TO EPROM
B0F2 BD  F8B3     A        JSR    OUT2HS    DISPLAY 2 HEX CHARS OF DATA
B0F5 BD  B209     A        JSR    SETADD    GOSUB TO SET EPROM ADDRESS
B0F8 8A  C0       A        ORAA   #%11010000 SET IDLE PGM MODE +25V & 5V
B0FA BD  B1BA     A        JSR    STROBE    GOSUB TO STROBE IDLE MODE INTO LATCH
B0FD 7D  F009     A        TST    TEMPAR    CHK FOR 1ST PROGRAMMING BYTE?
B100 27  06  B108          BEQ    STPRGM    PROCEED IF NOT...BUT
B102 7F  F009     A        CLR    TEMPAR    OTHERWISE, RESET FLAG &
B105 BD  B1EB     A        JSR    DELAY     DELAY TO DELAY WHILE +25V STABILIZES...
                   * START PROGRAM PULSE:-
B108 B6  F7FC     A STRPGM LDAA   DRA
B10B 7D  F008     A        TST    FRMCNT    CHK 2516 OR 2532 SELECTED?
B10E 26  04  B114          BNE    SET1AA    PROCEED TO 2516 SETUP IF FLAG NON-ZERO
B110 84  BF       A        ANDA   #%10111111 ELSE SET PGM BIT LO FOR 2532 $
B112 20  02  B116          BRA    CON1AA    GOTO START PGM PULSE DELAY...
B114 8A  20       A SET1AA ORAA   #%00100000 SET PGM PULSE BIT HI (2516)
B116 BD  B1BA     A CON1AA JSR    STROBE    LATCH BITS INTO DRA VIA CB2
B119 BD  B1EB     A        JSR    DELAY     FOR 50 MSEC....
B11C B6  F7FC     A        LDAA   DRA
B11F 7D  F008     A        TST    FRMCNT    CHK FOR EITHER 2516 OR 2532 SELECTED $
B122 26  04  B128          BNE    SET1AB    GOTO RESET 2516 PGM PULSE IF REQ'D...ELSE
B124 8A  40       A        ORAA   #%01000000 RESET 2532 PGM PULSE TO IDLE HI STATE  1
B126 20  02  B12A          BRA    CON1AB    PROCEED TO LATCH BITS INTO PIA...
B128 84  DF       A SET1AB ANDA   #%11011111 SET PGM PULSE BIT LO
B12A BD  B1BA     A CON1AB JSR    STROBE    LATCH BITS INTO DRA VIA CB2
                   * END PROGRAM PULSE:-
B12D BD  B222     A        JSR    SETIMP    GOSUB TO SET READ MODE
B130 B6  F7FC     A        LDAA   DRA
B133 84  0F       A        ANDA   #%00001111
B135 8A  10       A        ORAA   #%00010000 SET READ MODE W/ ONLY 5V ACTIVE
B137 BD  B1BA     A        JSR    STROBE    GOSUB TO LATCH READ MODE BITS
B13A CE  1000     A        LDX    #$1000    GET 20MSEC DELAY COUNT (1.5 MHZ) $
B13D BD  B1EE     A        JSR    DELOOP    DOES WAIT & 25V DEACTIVATES FOR READ (REQ'D BY 2532)
B140 F6  F7FE     A        LDAB   DRB       READ DATA BYTE FROM EPROM
B143 FE  F000     A        LDX    STRADR    POINT TO CURRENT RAM DATA
B146 A6  00       A        LDAA   0,X       GET RAM DATA BYTE
B148 11                    CBA              VERIFY READ/WRITE DATA?
B149 26  5A  B1A8          BNE    ERROR     IF NOT....BUT
B14B 39                    RTS              OTHERWISE, RETURN TO MLP...
                   ******** * ****************************************************************

B14F BD  B177     A VRFY   JSR    READ1     SET ADDR & READ EPROM BYTE
B152 FE  F000     A        LDX    STRADR    POINT TO CURRENT RAM
B155 E6  00       A        LDAB   0,X       GET RAM BYTE
B157 11                    CBA              VERIFICATION?
B158 27  16  B173          BEQ    DISOK1    GOTO DISPLAY 1 BYTE IF SO...BUT
B15D 7C  F007     A        INC    TEMPXR+1  OTHERWISE, BUMP LO-ORDER DISCREPANCY COUNT
B160 26  03  B165          BNE    SAVEDT    PROCEED TO DISPLAY DATA IF NO OVERFLOW...ELSE
B162 7C  F006     A        INC    TEMPXR    BUMP THE HI-ORDER COUNT TOO
B165 F7  F004     A SAVEDT STAB   ACCTRL    STORE RAM DATA
B168 B7  F005     A        STAA   ACCTRL+1  STORE EPROM DATA
B16B CE  F004     A        LDX    #ACCTRL
B16E BD  F8B1     A        JSR    OUT4HS    DISPLAY BOTH DATA BYTES
B171 20  03  B176          BRA    RTNVRD
B173 BD  [illegible]     A DISOK1 JSR    OUT2HS    DISPLAY OK BYTE
B176 39                  RTNVRD RTS

                   ***************************************************************************

B177 BD  F01C     A READ1  JSR    INCHK     KYBD CHAR WAITING?
B17A 27  11  B18D          BEQ    CONTAD    PROCEED IF NOT...BUT
B17C BD  F8B3     A        JSR    INEEE     OTHERWISE, GO GET CHAR
B17F 81  03       A        CMPA   #$03      CTRL/C?
B181 26  0A  B18D          BNE    CONRAD    PROCEED IF NOT...BUT
B183 BD  F833     A        JSR    INEEE     OTHERWISE, WAIT FOR ANOTHER CHAR...
B186 81  03       A        CMPA   #$03      CTRL/C?
B188 27  03  B18D          BEQ    CONRAD    CONTINUE IF SO...BUT
B18A 7E  B01A     A        JMP    IDLE      OTHERWISE, GOTO RESTART IN IDLE MODE...
B18D BD  B1DC     A CONRAD JSR    DSPLAY    ADDR NOW IF REQ'D
B190 BD  B222     A        JSR    SETIMP    GOTO SETUP INPUT LINES
B193 BD  B209     A        JSR    SETADD    GOTO LATCH EPROM ADDR
B196 8A  10       A        ORAA   #%00010000 SET 5V ONLY
B198 BD  B1BA     A        JSR    STROBE    LATCH READ MODE
B19B B6  F7FE     A        LDAA   DRB       READ EPROM BYTE
B19E 39                    RTS
B19F FE  F000     A READ2  LDX    STRADR    POINT TO CURRENT RAM
B1A2 A7  00       A        STAA   0,X       STORE EPROM DATA
B1A4 BD  [illegible]     A        JSR    OUT2HS    DISPLAY BYTE
B1A7 39                    RTS

                   ***************************************************************************

                   * MINOR SUBROUTINES:                                                      *
B1A8 36  02              ERROR  PSHA             SAVE RAM DATA
B1A9 4F                    CLRA             SET IDLE MODE
B1AA BD  B1BA     A        JSR    STROBE    GOSUB TO STROBE IDLE MODE
B1AD CE  B[illegible]     A        LDX    #ERRMSG   POINT TO STRING
B1B0 BD  F[illegible]     A        JSR    PDATA     GOSUB TO DISPLAY MESSG
B1B3 32  02                PULA             RESTORE RAM DATA
B1B4 FE  F000     A        LDX    STRADR    GET MEM ADDR $
B1B7 3F                    SWI              STOP:ACCA=RAM DATA  ACCB=EPROM DATA  XREG=FAILING EPROM AD

B1BA B7  F7FC     A STROBE STAA   DRA       STORE CTRL/ADDR BITS
B1BD B6  F7FF     A        LDAA   CRB       GET CURRENT CTRL REG B CONTENTS
B1C0 8A  08       A        ORAA   #%00001000 SET CB2 BIT HI
B1C2 B7  F7FF     A        STAA   CRB
B1C5 84  F7       A        ANDA   #%11110111 THEN SET CB2 BIT LO-STROBE LATCH
B1C7 B7  F7FF     A        STAA   CRB
B1CA 39                    RTS

B1CB CE  B37F     A GETRDY LDX    #READY    POINT TO INSERT EPROM MESSG
B1CE BD  F850     A        JSR    PDATA     DISPLAY MESSG $
B1D1 BD  F833     A        JSR    INEEE     GET CHAR FROM KYBD
B1D4 81  0D       A        CMPA   #$0D      RETURN KEY?
B1D6 27  0F  B1E7          BEQ    RTGRDY    RETURN TO CALLER IF SO...BUT
B1D8 7E  B01A     A        JMP    INSTRT    OTHERWISE GOTO SELECTION MODE...
```

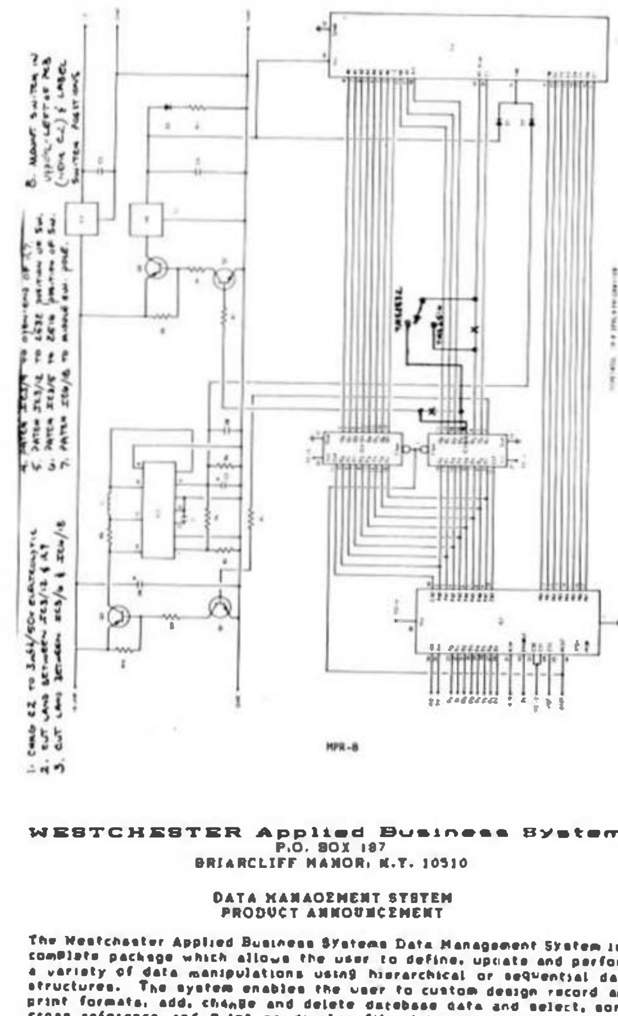
```
283A B1D0 39            DISP01 RTS
284
285A B1DC 7D    F00A    A DSPLAY TST     BYTECT  CHK CHARS PER DISPLAY LINE
286A B1DF 26    09 B1EA          BNE     DISP01  & RETURN TO CALLER IF NOT ZERO
287A B1E1 BD    F05A    A        JSR     PCRLF   OTHERWISE, SEND CRLF TO SCREEN
288A B1E4 BE    F000    A        LDX     STRADR  THEN POINT TO CURRENT MEM ADDR
289A B1E7 BD    F0B1    A        JSR     OUT4HS  & DISPLAY MEM ADR AS START OF NEW LINE
290A B1EA 39            DISP01 RTS
291
292D B1EB 0E    2300    A DELAY  LDX     #$2300  SET 1.5mS DELAY COUNT
293A B1EE 30    1F      DELOOP   DEX
294A B1F0 26    FC B1EE          BNE     DELOOP  FOR DEFAULT 50 MSEC DELAY
295A B1F2 39                     RTS
296
297A B1F3 7C    F00A    A BMPCHR INC     BYTECT  BUMP CHAR COUNTER
298A B1F6 86    10      A        LDAA    #$10    GET MAX PER LINE
299A B1F8 B1    F00A    A        CMPA    BYTECT  ONE DISPLAY LINE COMPLETE?
300A B1FB 26    03 B200          BNE     CONT1   IF NOT, CONTINUE...BUT
301A B1FD 7F    F00A    A        CLR     BYTECT  OTHERWISE, RESET COUNTER
302A B200 BE    F000    A CONT1  LDX     STRADR  GET CURRENT ADDR
303A B203 30    01               INX             BUMP BY 1
304A B205 BF    F000    A        STX     STRADR  UPDATE DYNAMIC POINTER
305A B208 39                     RTS
306
307A B209 B6    F001    A SETADR LDAA    STRADR+1 GET LO-ADDR BYTE
308A B20C B7    F7FC    A        STAA    ORA     STORE LO-ADDR IN PIA 1
309A B20F B6    F7FD    A        LDAA    CRA
310A B212 8A    08      A        ORAA    #%00001000
311A B214 B7    F7FD    A        STAA    CRA     BRING CA2 HI (HOLD)
312A B217 84    F7      A        ANDA    #%11110111
313A B219 B7    F7FD    A        STAA    CRA     BRING CA2 BACK LO=STROBE/LATCH LO ADDR
314A B21C B6    F000    A        LDAA    STRADR  GET HI ADDR BYTE
315A B21F 84    0F      A        ANDA    #%00001111 MASK OUT HI-NIBBLE
316A B221 39                     RTS
317
318A B222 B6    F7FF    A SETINP LDAA    CRB
319A B225 84    FB      A        ANDA    #%11111011
320A B227 B7    F7FF    A        STAA    CRB     SELECT DIR REG B
321A B22A 7F    F7FE    A        CLR     DRB     SET B LINES AS I/P
322A B22D 8A    04      A        ORAA    #%00000100
323A B22F B7    F7FF    A        STAA    CRB     SELECT DATA REG B
324A B232 39                     RTS
325
326A B233       0D      A SELECT FCB     $0D,$0A,$0A
327A B236       45      A        FCC     /ENTER SELECTION NUMBER: /
328A B24E       04      A        FCB     $04
329
330                       ********************************************************
331
332
333A B24F       0D      A TITLE  FCB     $0D,$0A,$0A
334A B252       43      A        FCC     /CDS 2516/2532 PROGRAMMING ROUTINE/
335A B274       0D      A        FCB     $0D,$0A,$0A
336A B277       31      A        FCC     /1.  PROG 2516 EPROM FROM RAM SOURCE/
337A B298       0D      A        FCB     $0D,$0A
338A B29C       32      A        FCC     /2.  PROG 2532 EPROM FROM RAM SOURCE/
339A B2BF       0D      A        FCB     $0D,$0A
340A B2C1       33      A        FCC     /3.  VERIFY EPROM AGAINST RAM SOURCE/
341A B2E6       0D      A        FCB     $0D,$0A
342A B2E8       34      A        FCC     /4.  READ EPROM DATA IN @ RAM SOURCE/
343A B309       04      A        FCB     $04
344
345A B30A       0D      A PARAMS FCB     $0D,$0A,$0A
346A B30D       43      A        FCC     /EPROM ADDRESSES:/
347A B31D       0D      A        FCB     $0D,$0A
348A B31F       20      A        FCC     /   2516: $0000-$07FF/
349A B333       0D      A        FCB     $0D,$0A
350A B335       20      A        FCC     /   2532: $0000-$0FFF/
351A B349       0D      A        FCB     $0D,$0A,$0A
352A B34C       45      A        FCC     /ENTER:/
353A B351       0D      A        FCB     $0D,$0A
354A B353       20      A        FCC     /   STARTING ADDRESS:/
355A B367       04      A        FCB     $04
356A B368       0D      A ENDA   FCB     $0D,$0A
357A B36A       20      A        FCC     /   ENDING   ADDRESS:/
358A B37E       04      A        FCB     $04
359
360A B37F       0D      A READY  FCB     $0D,$0A
361A B381       53      A        FCC     /SWITCH-SELECT 2516 OR 2532/
362A B39C       0D      A        FCB     $0D,$0A
363A B39E       49      A        FCC     /INSERT EPROM & PRESS 'RETURN' KEY:/
364A B3C0       04      A        FCB     $04
365A B3C1       0D      A PROCOM FCB     $0D,$0A
366A B3C3       50      A        FCC     /PROGRAMMING COMPLETED/
367A B3D8       0D      A        FCB     $0D,$0D,$04
368
369A B3DB       0D      A VERCOM FCB     $0D,$0A
370A B3DD       56      A        FCC     /VERIFICATION COMPLETED/
371A B3F3       0D      A        FCB     $0D,$0D,$04
372
373A B3F6       0D      A RDDON  FCB     $0D,$0A
374A B3F8       52      A        FCC     /READ COMPLETED/
375A B406       0D      A        FCB     $0D,$0D,$04
376
377A B409       0D      A ERRMSG FCB     $0D,$0A
378A B40B       56      A        FCC     /VERIFICATION ERROR: /
379A B41F       0D      A        FCB     $0D,$0A
380A B421       20      A        FCC     /   MEM ADDR    EXPECTED DATA    EPROM DATA/
381A B454       04      A        FCB     $04
382
383A B455       0D      A VERMSG FCB     $0D,$0A
384A B457       21      A        FCC     /NO DISCREPANCIES -$$/
385A B46C       04      A        FCB     $04
386
387                       ********************************************************
388
389                       ********************************************************
390
391             B000    A        END     EPROMS
```

```
No Errors detected during this assembly
No Warnings noted during this assembly

F004 ACCTRL    188  225  226  227
FB6D BADDR      58   94   98
B1F3 BMPCHR    122  134  154  297A
F00A BYTECT    229   86  285  297  299  301
B18D CHREAD    236  239  262  260A
B116 CON16A    190  179A
B12A CON16B    198  200A
B200 CONT1     300  302A
B075 CONTPG    123A 124
B08E CONTRD    152  156
B071 CONTVF    113A 136
F7FD CRA       308   71   77  309  311  313
F7FF CRB       400   72   70  167  300  173  270  272  274  310  320  323
B1EB DELAY     184  193  292A
B1EE DELOOP    208  293A 294
B173 DISBYT    221  230A
B1EA DISP01    278   30   39   80   74  186  0F0  283  267  300
F7FE DRB       370   75  171  176  209  349  321
B1DC DSPLAY    166  260  285A
B368 ENDA       70  356A
F002 ENDADR    178  100  133  135  155
B000 EPROMS    718  391
B409 ERRMSG    263  377A
B1A8 ERROR     213  260A
F008 FRMCNT    200  117  197  195
B034 GETPRM    910
B1CB GETRDY    120  130  131  277A
B010 IDLE        8  117  147  159  263

F04C INCH       60  135
FB33 INEEE      70   83  237  240  279
B01A INSTRT    010   80   90  110  202
B0AC KEYVFY    140  143A
F0B3 OUT2HS     98  177  230  253
F0B1 OUT4HS    100  146  230  209
B30A PARAMS     92  345A
F05A PCRLF     118  287
F0BE PDATA1     68   82   84   93   97  126  130  144  150  264  270
B3C1 PROCOM    125  365A
B00D PROG      171  166A
B064 PROG32    103  114A
B06B PROGRM    105  116A
B177 READ1     152  217  225A
B19F READ2     153  251A
B08B READOP    109  151A
B37F READY     292  360A
B3F6 RDDON          373A
B1CA RTNDIS    206  290A
B1D8 RTNRDY    201  283A
B0B8 RTNVER    142  147A
B176 RTNVRF    229  231A
B165 SAVEBT    223  223A
B233 SELECT     83  326A
B114 SET16A    180  191A
B128 SET16B    196  199A
B209 SETADR    170  246  307A
B222 SETINP    202  243  318A
B0A7 STPROG    135  137A

F000 STRADR    160   95  174  310  218  281  266  288  302    8  307  314
B1BA STROBE     80  180  192  200  204    2  242  269A
B100 STRPGM    182  186A
F009 TEMPAR    218  118  119  181  183
F006 TEMPXR    170  132  130  161  165  222  224
B24F TITLE      83  333A
B3DB VERCOM    137  369A
B08E VERIFY    107  130A
B455 VERMSG    143  383A
B1A0 VRFY      133  212A
```



MPR-8

WESTCHESTER Applied Business Systems
P.O. BOX 187
BRIARCLIFF MANOR, N.Y. 10510

DATA MANAGEMENT SYSTEM
PRODUCT ANNOUNCEMENT

The Westchester Applied Business Systems Data Management System is a complete package which allows the user to define, update and perform a variety of data manipulations using hierarchical or sequential data structures. The system enables the user to custom design record and print formats, add, change and delete database data and select, sort, cross-reference and print or display file data on the terminal. It is an extremely powerful system derived from a number of commercial vendor offerings, capable of stand-alone usage for a wide variety of home and business applications.

Typical examples of use are schedules, customer data, checkbooks, addresses, inventory, records retention, market trends, mailing lists and most other application requiring the structuring of data. The DMS system, when properly applied, can displace the need for many specialized programs and, thus, reduce long term software costs.

Two DMS systems are available. DMS1 is an entry level system which supports file sizes up to 32K and DMS2/VM is a full scale business oriented system which permits file sizes up to 1000K. The basic functions of both systems are identical. Both DMS1 and DMS2/VM systems consist of the following programs:

The DEFINE program is the facility by which the user identifies the format and content of a database. Field name, format and size and groups of fields are identified to the system, and the file definition becomes a permanent part of the database. This enables later reference and titling of the data by name. The structure may be hierarchical, which reduces redundancy, and up to 25 field or group definitions are permitted. Alphanumeric, numeric, decimal, integer, coded and hexidecimal field types are supported. Since the definition is part of the file, and not part of an external data dictionary, the files may be manipulated by standard utility commands such as COPY, DELETE and RENAME.

The UPDATE program is a facility enabling the user to review, input, add, change and delete database information. These editing functions are executed via the system terminal on an interactive basis. Single character commands facilitate execution of desired update functions with minimal effort.

The GENER program is the real power of the DMS system. This program accepts English-like instructions from the terminal and produces reports, displays on the terminal screen or generates new databases per user command. The GENER program may be applied to selection, inquiry, analysis and report applications eliminating the need for specific programs. The commands may be stored in a file, reducing the execution of complex functions to the entry of two words!

GENER functions available include match and range selection of input records on multiple fields, extraction of specified fields, lookup of data on other databases via multiple keys (without pre-defining record indexes), array generation and summation, sort by multiple fields and formatting of output on the terminal, printer or disk. GENER command files may set up using symbolic parameters, for which the user is prompted at execution time, enabling inquiry applications. GENER disk outputs may be reread as input so that batch-like operations may be run for complex processing requirements.

The FORMAT program enables the user to custom design output print forms using database information. The program employs a user supplied control file containing titles, text, graphics and/or field names which, in print form, resembles the desired output. This file is used as a "mask" to format the database fields specified. Field and mask data may appear anywhere on the printed page, and the mask may be of any size enabling use of the FORMAT program for mailing labels, contracts, bills, letters, preprinted forms and other applications.

Included with the DMS system is an OUTPUT program. This program enables the user to direct output from other programs to the printer, terminal screen or to disk. The program also supports the variable format capabilities of the Centronics 737 dot matrix printer.

At the heart of the DMS system is the resident DMS "nucleus" module. This program contains over 200 arithmetic, data manipulation, disk, printer and terminal I/O functions, many of which emulate instructions found in commercial maxi-computers. These functions afford maximum flexibility and standardization, and substantially reduce the size of programs running under DMS. This results in an increase of memory space available for user data and virtually bug-free programs.

DMS2/VM employs virtual memory techniques to increase the effective memory available for user data. This methodology utilizes an entire disk drive as a direct access "paging device" to swap in and out sections of memory as they are required. The result of this is the ability to direct access files in excess of 1 megabyte. The feature is invoked only as needed so that normal operations using small files may utilize both disk drives. A VMGEN program is included with the DMS2/VM system and is used to format the paging disk.

The DMS systems are fully compatible with our accounting system and other database oriented software, allowing specialized accounting and inventory reports and data manipulation per user requirements.

The DMS1 system sells for $250.00 and the DMS2/VM system for $650.00. Both systems require a license agreement and forms are available from WESTCHESTER Applied Business Systems. The 32 page user guide is also available for $10.00, and is deductible from system purchase.

J. J. GLIDEWELL
GIMIX 6800/6809 Computers
3623 Charlene Drive
Dayton, Ohio 45432
(513) 426-3887

Mr. Don Williams
'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixon, Tenn 37343

ANNOUNCEMENT November 28, 1980

I am pleased to announce the opening of my business for sales and support of SS-50 Bus products and computer systems. I will be available to help you in any way I can, so contact me about your requirements or any questions you may have. I can meet your needs with the complete line of GIMIX products as well as for printers and 5 inch disc drives. Call or write to me at the address below.

J. J. Glidewell, 3623 Charlene Dr., Dayton, OH 45432

John Glidewell

When arriving in Chicago after a 29 hour flight from Australia, the last thing I wanted to do was go to a party being held at the Don's residence.

This party was loaded with all weird and wonderfull people, which brought me closer to the 6800 Bus scene. This party was the coming together of National and International dealers, manufacturers, programmers and the press.

Monday the 4th of May the NCC Show opened it's doors. Exhibitors from the biggest, IBM, down to Micro manufacturers showed their wares.

GIMIX had a booth at which they had five different mainframes. On one, Ray Talbott demonstrated Forth on the 512x512 Graphics screen. Ken Kaplan of Microware had OS9 running on 3 terminals, one of which was in colour. Sam Epstein had a mind blowing demo program running in colour. I really liked using BASIC 09 and noticed his Pascal also running. Dave Allen from Tallgrass Technologies had UCSD Pascal.

Wandering around the McCormick Hall Radio Shack was showing off their latest TRS-80C colour computer which was rather impressive. Commodore also had their new Dual Processor micro which had a 6809 and a 6502.

Meanwhile back at the GIMIX booth, Richard Don and Bob Phillips were demonstrating some new goodies. I saw their new DMA disk controller which runs 5" or 8" disk drives, and also had a good look at their new 64K Static RAM memory card with battery back up. It uses 6116 CMOS 200ns. chips. A Winchester 3.3 Mbytes drive was running on the GIMIX system.

Tom Crosley of Programma International gave me lessons on how to use the PIE text editor. (This report was written using PIE). Heinrich Zacher of West Germany and Willi Vollenweider of Switzerland compared notes on the different Pascal Systems available for the GIMIX 6809 systems. 7 different Pascals were available: UCSD from Tallgrass; Microware; TSC's; Omegasoft; Lucidata; Kenyon Microsystems Forth version; and Al Jost flew in from Nova Scotia to show for the first time the Dynasoft disk version.

Ron Anderson (Flex User Notes writer), Shawn Morrissey and Randy Lewis from Ann Arbor visited the GIMIX booth. Bob Bundy of Sohoe, from New York, demonstrated Stylograph, a word processing editor. Jerry Koppel from AAA Chicago Computer Center demoed his Payroll Package.

Don and Joyce Williams made their presence known for most of the show. Charlie Williams, another member of the family, was left at the hotel to deal with important business matters concerning 68 Micro Journal.

I spent a whole night listening to Les Solomon cracking jokes and swapping stories with my dad, Paris Cockinos, and Richard Don. Les Solomon started the micro industry with his article in Popular Electronics about the Altair. The SOL computer was also his brainchild and was named after him.

SWTPC was represented at NCC by Semiconductor Specialities which displayed the 8212M, a new Word Processing terminal. SSB, Tano and Winter were also there among the other 6800 & 6809 exhibitors.

The 1981 NCC had an attendance of over 73,000, not including exhibitors.

Jacky Cockinos
Paris Radio and Electronics
P.O. Box 380, Darlinghurst 2010
Sydney, NSW, Australia

TEXAS TECH UNIVERSITY HEALTH SCIENCES CENTER
SCHOOL OF MEDICINE, Department of Biomedical Engineering & Computer Medicine
Lubbock, Texas 79430 (806) 743-2787

April 28, 1981

Editor, '68 Micro Journal
3018 Hamill Rd.
P.O. Box 849
Hixson, Tennessee 37343

Dear Sirs:

I have interfaced an EPSON MX-80 printer to a SWTP S09 system using the MP-ID parallel port. Since this is a popular printer, I offer the following set up for your readers.

```
MP-ID connector                        MX-80 parallel
   14 ---------------------------------- 2
   15 ---------------------------------- 3
   16 ---------------------------------- 4
   17 ---------------------------------- 5
   18 ---------------------------------- 6
   19 ---------------------------------- 7
   20 ---------------------------------- 8
   21 ---------------------------------- 9
   22 ---------------------------------- 1
   23 ---------------------------------- 10
    1 ---> - - - - - - - - 16, 19, 20, 21, 22, 23, 24, 25, 26,
       L---> - - - - - - - 27, 28, 36
                       connect 14 to 35 on the MX-80 side
```

On the MX-80 side, data lines are 2 thru 9. Line 1 is the strobe input which is effectively generated by line 22 of the MP-ID. Line 14 on the MX-80 must be tied high by line 35 on the MX-80 or double spacing occurs. The other lines listed are tied to signal ground. All other lines can be ignored.

In the MX-80 are two sets of switches. They should be set as follows.

Switch #	SW2 ON	SW2 OFF	SW1 ON	SW1 OFF
8			X	
7				X
6			X	
5			X	
4		X	X	
3		X		X
2	X		X	
1	X		X	

Sincerely yours,

Blair A. Rowley, Ph.D., P.E., C.C.E.
Professor and Chairman

BAR/sm

Mr. Don Williams
68 Micro Journal
3018 Hamill Rd
Hixson, Tennessee 37343

15 May 1981

Dear Don,

I had never expected such a wide audience for my performance grumbles! But from the comments and article on the April and May issues I believe some people didn't get the right point. Exception only to Ray Talbot with his FORTH measured to, which are done on the same program, same machine speed and size as the ones I had done in my March article.

I want to address some of the criticisms voiced, in particular Dave Shirk's article. I never intended to critisize TSC or their software, but to compare the few alternatives FLEX users have today as far as programming languages are concerned. I also never pretended to present any best method to compute prime numbers (it could have been any program really) or see how fast a specific problem can be made to run on a 6809. My interest was to compare real software packages running the same algorithm (in a way, the more inefficient the better because the more machine resources it will use and therefore show a reasonable picture of a heavily used system) and get a bit more insight on the relative speed of each.

My assumptions were simple: a single user, FLEX 9, 56K and 1 MHz. At this side of the ocean there is not much choice if you need a high level language under FLEX, you either choose one of the TSC Basics or Lucidata PASCAL. Although from a purist technical point of view comparing integer and real arithmetic languages may seem strange, it is a very valid comparison from the practical point of view: many of us will be faced with the TSC Basic vs. Lucidata Pascal choice if we want to use a high level language under FLEX. If somebody thinks this is an unfair comparison, please supply a Basic that handles integer arithmetic on a 56K FLEX system and is faster than Lucidata Pascal on the same problem.

To compare FLEX and UNIFLEX software as Dave Shirk did (April issue, page 13) is unfair. UNIFLEX does not run on a 56K machine. It is not an option unless you have 128K and are prepared to pay twice the price of a 56K machine. Yet Tom Herman pointed out (May issue, page 18) that BASIC09 gives a performance comparable to the UNIFLEX Basic and yet does not need more than 56K to run. The point here is that we smaller users are been left behind for bigger machines and more expensive systems and I dont think it is fair. Another point is that I would appreciate if Dave Shirk had stated the clock speed of his system: if his measurements are not made in a 1 MHz system then one should multiply the timings by the corresponding factor. Finally, although I admit that nice features like position independence and reentrancy are useful, they are not that relevant to us single users with not so big machines; but I really get bothered when my SWTP C/09 takes 4 or 5 hours to chug across some big program.

A final word: take a look at Ray Talbot's timings (May issue, page 27) for FORTH. With a 56K FLEX 1MHz single user machine (the same as my machine) and using exactly the same algorithm as I did, his FORTH program runs heaps faster than UNIFLEX Pascal and just about as fast as the improved algorithm runs. Take into account that FORTH is an interpreted language while Pascal is a compiled language, and you see that compiled code ain't necessarily faster than interpreted code. Keep it up, Ray - I think you scored a point here.

Al Moreira
22 The Paddock
Chalfont St Peter
Bucks SL9 0JB
ENGLAND

FLEX and UNIFLEX are trade marks of TSC Inc. BASIC09 is a trademark of Microware.

May 13, 1981
406 40th Street
New Orleans, LA 70124

Don Williams
'68' Micro Journal
3018 Hamill Road
P.O. Box 849
Hixson, Tennessee 37343

Dear Don,

Some of your readers with TANO Corporation Outposts who have switched from TANO BASIC to Technical Systems Consultants BASIC undoubtebly miss the direct cursor positioning of TANO BASIC. The following TSC BASIC subroutine will print the string A$ at column C, row R, where column 0, row 0 is the upper left hand corner of the CRT:

```
9000 PRINT CHR$(28);
9010 IF PEEK(65072)<128 THEN 9010
9020 POKE 65074,C
9030 POKE 65075,R
9040 PRINT CHR$(28);
9050 PRINT A$;
9060 RETURN
```

I hope that this subroutine solves a problem for some of your readers.

Sincerely,

James L. Dean

Mr. Don Williams, Sr. - Publisher
'68' Micro Journal
P. O. Box 3018 Hamill Road
Hixson, Tennessee 37343

John C. Tarvin
[illegible] Ct.
[illegible] Hill, California [illegible]
(408) [illegible]

Dear Don :

Thanks for your letter of 3/30/81. I am looking forward to the May issue and the chance to see some of the contest software. I'll bet there are some real goodies there.

Don, I have SWTPC version 2.6 of Flex 09 and it has the PSP.CMD in it. I have noted a discrepency in the instructions. One place it indicated to put the MP-T card in I/O port 5 and in another it said I/O port 4. Needless to say I have tried both, but to no avail. Seems the first thing that happens when I try to spool is I get a message that there is insufficient memory for the printer driver. If I use the RM.CMD simply to see what is available, it indicates 512 bytes (enough). When I then try PSP.CMD, it will load the .OUT files into the queue, but alas, no printing occurs. As suggested by the errata sheet with the SP-09 CPU, I configured SBUG for no timer. I also tried it with timer=yes. I am buffaloed. I questioned the MP-T card, but it seems to work fine.

I have sold my two MPA2 cards so there is no need for you to run the classified. Thanks anyway. I think my next project will be to upgrade to the GIMIX 6809 CPU. Do you run one of those? If so, how do you like it?

Seems there is a 68XX(X) users group here in the Bay Area. I just found them and the next meeting is 14 April. I plan to attend and maybe I can find a solution to my spooling problem.

I know you are busy, so won't take any more of your time. Can't tell you how much I enjoy the Journal. Take care and I'll contribute some more software soon.

Sincerely yours,

John C. Tarvin, ESSMC

HELP

DOES ANYONE HAVE THE CAPABILITY OF TRANSFERRING TAPE DATA TO SWTPC'S DMF-2 DISKETTES? TAPES ARE 9 TRACKS,1600 BPI,UNLABELED WITH BLOCKING FACTORS OF 1 X 186 AND 1 X 286.CALL ME COLLECT AT 214-276-6196 IF YOU CAN TRANSFER THE DATA OR KNOW ANYONE WHO CAN. LARRY WOLFE, 519 STATE ST,GARLAND,TX 75040

CLASSIFIED

FOR SALE,SWPTC 6800 COMPUTER 28K PR40 PRINTER-7MP BOARDS-CT 64 TERM- SANYO MON-AC30 & CIS30+INTERFACES EXTRAS-CPU BOARD W/Z80 CHIP,6809 CPU BOARD(PERCOM),SWTBUG-RT68 MON.S32 BLANK MEMORY BOARD(SWPTC) BEST OFFER OVER $600.00 301 367 2760 AFTER 5PM C N HARRID POB 3021 BALTO, MD 21229

*****

SWTPC 6800-2 MICROCOMPUTER SYSTEM A&T/SOCKETS 24K RAM, 3 SERIAL & 1 PARALLEL I/O BOARDS, EPROM PROGRAMMING BOARD, JPC 4800 BAUD CASSETTE INTERFACE. THOMAS VIDEO I/O BD, HUMBUG + CFM-3 MONITORS. 16K RAM & 32K RAM/EPROM BOARDS A&T LESS MEMORY. EXTENSIVE HARD,SOFTWARE DOC'S. 3 BASICS, ASSEMBLER/EDITOR AND MORE COMPLETE SYSTEM PROFESSIONALLY PACKAGED WITH P.S., 9" CRT & ASCII KEYBOARD INTO ATTRACTIVE ENCLOSURE FOR PORTABILITY, $995.00 COMPLETE OR SELL ITEMS SEPERATELY, LET'S TALK (617) 935-3758 AFT 6 PM & WKENDS. PAUL.

*****

2 SWTPC 6800'S EACH WITH 24K, MPC, MPS, MP-L,JPC CLOCK 4800 BAUD JPC, 2 CASSETTE DECKS, CT-64, SANYO 9", FAST BASIC W/FILES ASSEM.,DIS, MUCH SOFTWARE. $800 EACH OR $1500 FOR BOTH WITH PR-40 PRINTER! MAINE COMPUTER NETWORK (207) 244-7444

*****

FOR SALE: JPC TC-3/CFM-3 EPROM $50, MPA $55, AC-30 $40, DSD-32K EPROM BOARD $50, SMITH BUG EPROM $30, 4K SWTP RAM $35 EACH, F&D CLOCK $65, RT-68/MX ROM $25, MINIFLEX $20, 6800 SOFTWARE. NEED MP-09 BAREBOARD OR ASSEMBLED. WILL CONSIDER TRADE. (606) 269-5909 5-10 pm EST.

*****

ENGINEERING-MATHEMATICAL-CAD SOFTWARE I AM INTERESTED IN WRITING/MARKETING THIS VARIETY OF SOFTWARE. IF YOU HAVE NEEDS OR OFFERINGS IN THESE AREAS, PLEASE CONTACT: R. ROGERS, 2831 ATWELL AVE., OAKLAND, CAL 94601.(415)261-1350.

*****

OmegaSoft Pascal Products

* SYSTEMS PROGRAMMING: Our Editor, Assembler, and Linker are written in OmegaSoft Pascal making use of full string support
* MACHINE CONTROL: Features like assembly language interface, fixed and stacked variables, hex data type support, and romable native code make the OmegaSoft Pascal Compiler a natural for equipment and process monitoring and control
* APPLICATIONS PROGRAMMING: Inclusion of a symbolic debugger, comprehensive compilation and runtime error handling, and ease of interactive I/O aid in the development of applications programs

6809 Compiler package generates romable, position-independant object code. Version 1 doesn't include records or pointers, those will be available in Version 2 later this year (to upgrade will cost $55 to $95 depending on options). Minimum of 32K byte system required with either the MDOS (trademark of Motorola) or FLEX (trademark of TSC) operating system. Write or call for data sheet, ordering information, and license agreement.

PRICING: Compiler package including assembler, linker, and debugger ______ $215
for source of runtime add ______ $ 50
user manual seperately ______ $ 23

OmegaSoft
P.O. BOX 70265
Sunnyvale, CA 94086
(408) 733-6979

$50.00 EZDATA $50.00
MANAGEMENT SYSTEM

EASY TO CREATE
EASY TO MAINTAIN
EASY TO REPORT

True random access (FAST)
Max. records on file (65,535)
Runs in under 16K

6800 assembly code (FAST)
Full record/Screen updating
Change File sequence on demand
Sort/Print any item in any Sequence

Absolute Cursor Control
Insert A Character/Expand Field
Delete A Character/Contract Field
Screen Paging - 126 page Max.
Online/Real Time updating
Full Forward/Backward/ Search Control
Cursor stops ONLY on the item to be Added/Changed

All fields identified by USER defined Names.
(No more tran codes to memorize).

Interact with your data in a language we can all understand — ENGLISH —

All commented source Programs on Three (3) 5 1/4" or One (1) 8" Disc.

Included: EZCTL, EZDATA, EZTEXT, EZVIEW, EZEDIT, EZPRINT, EZPURG, EZSORT
Please specify SSB or FLEX and 5 1/4" or 8" Disc.

GARY A. MAGNUSEN
208 TINKLER
LAFAYETTE, INDIANA 47901
Evenings 1-317-742-1565

Flex is a Trademark of Techinical Systems Consultants.

VC-256
GRAPHICS

The VC-256 is a high resolution graphics interface for the SS-50 bus. The controller incorporates a variety of unique and innovative features which provide excellent display quality combined with EXTREME SIMPLICITY of use. It will drive any monitor with composite video input.

Featuring . . .

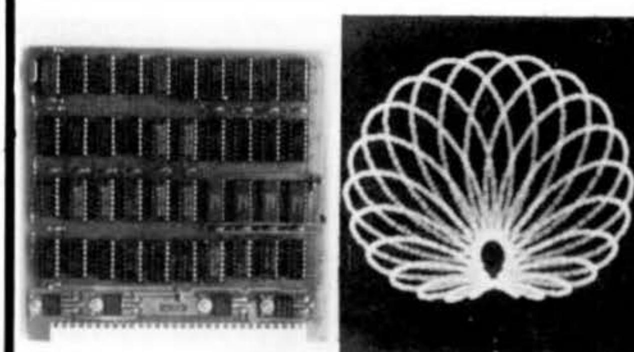
- individual pixel control
- true X-Y addressing
- single instruction erase
- independent blanking control
- jitter free display
- industrial quality construction
- fully socketed
- no system memory utilized
- no address space occupied
- no splatter on update
- no adjustments
- no software driver
- no software initialization
- no throughput loss

Specifications

Resolution	256 x 256 (256 x 250 on some monitors)
Bandwidth	8 MHz
Stability	crystal controlled
Addressing mode	X-Y single pixel
Origin	upper left corner
Writing rate	64 microseconds per pixel
Erase time	16.7 milliseconds
Write sync	interlocked
Blanking	program controlled
Output signal	non-interlaced composite video
Memory	65,536 bits in X-Y array on board
Registers	Write: X, Y, Z, Erase Read: status
Port addresses	4 in I/O address space
Physical location	one slot of 30 pin I/O bus
Size	5.6 in x 5.6 in
IC count	40 + 4 regulators
Output	75 ohm coax

SOFTWARE SUPPLIED
(6809 5¼" FLEX™) INCLUDES:
- Camera Digitizer Program
- Exerciser Program
- Character Generator Routine
- Misc. Pattern Programs
- Line Drawing Routine

(All with Source Code)



PRICE: $350 — assembled, tested, and burned in
AVAILABILITY: stock to 30 days WARRANTY: 90 days
Supplied with 6 feet of cable less video monitor connector.

GIMIX STOCKING DISTRIBUTOR

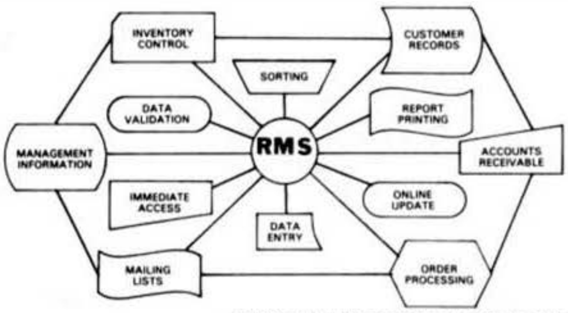
HAZELWOOD COMPUTER SYSTEMS
7413 NO. LINDBERGH, HAZELWOOD, MO 63042 (314) 837-3466
MasterCharge VISA American Express Diners Club
DEALER INQUIRIES INVITED

FLEX™ IS A TRADEMARK OF TECHNICAL SYSTEMS CONSULTANTS

6809 RECORD MANAGEMENT SYSTEM RMS

DATABASE MANAGEMENT

- USER DEFINED RECORD FORMAT VIA DATA DICTIONARY
- SCREEN ORIENTED, FORM FILL OUT TYPE OF ACCESS
- OPTIONAL TWO LEVEL RECORD HIERARCHY
- ALL FILES IN ASCII TEXT FORMAT, BASIC COMPATIBLE
- DIRECT ACCESS BY KEY FIELD, MULTIPLE INDEX FILES
- EXTENSIVE DOCUMENTATION, SAMPLE APPLICATION
- VERSATILE, PROFESSIONAL QUALITY REPORT WRITER
- BUILT-IN SORT / MERGE
- EASY TO USE



RMS is a complete DATABASE MANAGEMENT package for the 6809 computer. It is made up of five machine language programs that make up the most powerful business programming tool available for the 6809. It can be used by the relative novice, to implement an incredible variety of information storage and retrieval applications, without any programming. However, the programmer can use RMS as part of the solution to a larger problem, saving many hours of unnecessary program development time. RMS can be used to handle data input, editing, validation, on-line retrieval, sorting and printed reports. Custom data manipulation can be filled in by the user's BASIC programs.

SINGLE CPU LICENSE

FLEX*	$200
OS-9+	$250
UNIFLEX*	$300

TERMS: VISA / MC / PREPAID

WASHINGTON COMPUTER SERVICES
3028 SILVERN LANE
BELLINGHAM, WA 98225
1 (206) 734-8248

* FLEX and UNIFLEX are trademarks of Technical System Consultants Inc.; + OS-9 is a trademark of Microware

AT LAST... A top-notch assembly language programming text based on the Motorola 6809:

MICROCOMPUTER ARCHITECTURE AND PROGRAMMING

by John F. Wakerly
Stanford University

Published in March 1981
by John Wiley and Sons, Inc.

Here is a well-written, authoritative text on the basic principles of microcomputer organization and assembly language programming using the Motorola 6809 as the principal example. MICROCOMPUTER ARCHITECTURE AND PROGRAMMING:

- Develops general principles of microcomputer organization and programming, using the 6809 for assembly language programming examples (Chapters 5-12).
- Concisely describes high-level language programming in Pascal (Chapter 2).
- Discusses the important data structures needed in most applications (Chapter 3).
- Completely describes, from a programmer's point of view, seven important contemporary microprocessors, with special emphasis on 16-bit machines (Chapters 13-19).

Outstanding physical features of the book include:

- Over 700 pages packed with useful information
- 22 pages of indexes
- Over 170 illustrations
- Over 200 programming examples
- Over 80 tables

Here's what some knowledgeable reviewers had to say:

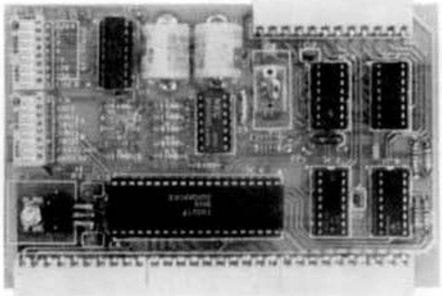
"This is an excellent manuscript! It is authoritative, well-organized, well-written, and very much up-to-date." — *David A. Hodges, Professor of Electrical Engineering, University of California, Berkeley*

"Looks great! It's nice to see someone who understands the 6809 and is so careful to get the details correct." — *Joel Boney, co-architect of the Motorola 6809*

TO ORDER: Use coupon in June issue or send $25.00 plus $2.00 postage and handling to **MSE Books, 257 Castro St., Suite 2E3, Mountain View, CA 94041.** Allow two weeks for delivery. Add $2.00 for air mail. Calif. residents add $1.50 sales tax. MasterCard and Visa welcomed; include card number and expiration date.

TABLE OF CONTENTS

CALENDAR-CLOCK / TIMER / PARALLEL PORT



Calendar - Clock CLK68-1

- [illegible]
- [illegible]
- [illegible]
- [illegible]

Interval Timer

- [illegible]
- [illegible]
- [illegible]
- [illegible]

Parallel I/O Port -- Fully buffered 8 bit parallel port

- [illegible]
- [illegible]

Construction -- [illegible]

Manual -- Well documented - 36 pages

Dealer & OEM discount available

Assembled and tested	$119.95	Kit	$89.95
Goldplated bus conn	7.50	2 MHz option	2.50
Disk 5 or 8 in. SSB or Flex* (OS-9** Avail soon)			14.95

** OS-9 is a trademark of Microware Systems Corporation
* Flex is a trademark of Technical Systems Consultants, Inc.

ROBERTSON ELECTRONICS
1003 Warm Sands Dr. SE
Albuquerque, NM 87123

Phone (505) 294-0025
NM residents add 4% tax
Add $3 Shipping & Handling

COMPUTER SYSTEMS CONSULTANTS, INC.

1454 Latta Lane, Conyers, GA. 30207
Telephone 404-483-1717 or 483-4570

SOFTWARE DEPARTMENT

(ALL PROGRAMS PROVIDED IN SOURCE ON DISK SPECIFY 5"/8")

SUPER SLEUTH Disassembler System (for FLEX* systems)		$ 99.00
—runs on 6800/1/9, analyzes 6800/1/6/9 and 6502		
—easy to use, self-instructive, with 42-page manual		
—automatic labels, optional FCB, FCC, FDB's		
—input binary file from disk or from memory		
—memory changes to program thru full-screen editor		
—output disk file may be source or new binary file		
—eo nds from enu or from and to disk file		
—generates FLEX* and user-defined names		
—includes assembler language XREF program		
—contact SMOKE or CER-COMP for non-FLEX* systems		
Z-80/8080/8085 Disassembler (Similar to SLEUTH)		$ 99.00
—runs on 6800/1/9, analyzes Z-80/8080/8085		
6800/1,6805, 6502,Z-80,8080/5 Cross-assemblers	EACH	$ 50.00
—macro sets for TSC 6809 Macro Assembler	ANY 3	$100.00
FULL-SCREEN FORMS DISPLAY for TSC 6809 X-BASIC		$ 50.00
—display and edit for terminals and video displa e		
—complete cursor control for screen input/output		
—interactive forms generator/documentor provided		
FULL SCREEN MAILING LIST System for TSC 6809 X-BASIC		$ 100.00
—full screen update and selection to print or labels		
TABULA RASA Tabular Calculation Program (COMING SOON)		$100.00
—similar to VISICALC (TM Personal Software)		
TSC BASIC Resequencing and XREF Programs		$ 25.00
—processes TSC BASIC, X-BASIC, PC, XPC programs		
—partial and blank-resequence capabilities		
—alphabetized xref of all variables and BASIC verbs		
TSC X-BASIC DISK SORT/MERGE Generator		$ 25.00
—generates TSC XPC BASIC sort/merge programs		

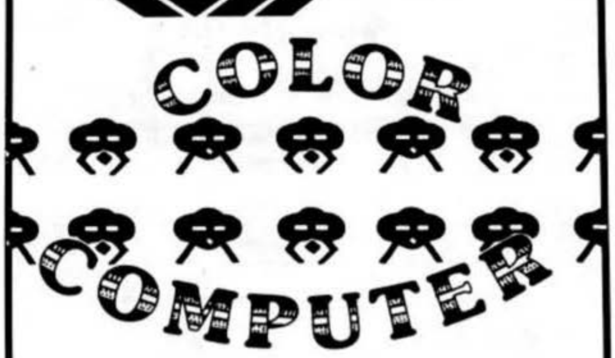
HARDWARE DEPARTMENT

(ALL BOARDS BARE WITH FULL DOCUMENTATION PROVIDED)

I/O SELECTRIC INTERFACE BOARD (serial or parallel)	$ 35.00
—ASCII (TTL or RS-232+CTS) in, 26-50v solenoids out	
—2708 PROM with Correspondence ball codes	$ 15.00
SS-50 WIRE-WRAP BOARD (52-16 pin equivalent)	$ 25.00
SS-30 WIRE-WRAP BOARD (32-16 pin equivalent)	$ 15.00
SS-30 DUAL ACIA BOARD (modem control + Baud rate gen)	$ 30.00
SS-50 FRONT PANEL DISPLAY BOARD (16 decoded LED's)	$ 10.00

VISA and MASTER CARD preferred: account, exp date, phone no.
US funds only. Add 7.5% (15% Foreign) for postage & handling
For Catalog or dealer discount information contact Bud Pass
*FLEX is a trademark of Technical Systems Consultants

COMPUTERWARE™



GRAPHIC GAMES Color Invaders • Action Games
• Asteroid Action • Adventures and More
PROGRAMMERS TOOLS Monitor • Assembler
• Editor • PASCAL • Diagnostics
HOME PROGRAMS Data Organizer
• Home Finance • Educational Programs
MEMORY • BOOKS • ACCESSORIES
Shipping from Stock

Call or Write:
COMPUTERWARE™
Dept. C • Box 668
Encinitas, CA 92024 • (714) 436-3512
6809 Specialists

Computerware is a trademark of Computerware

NEW! SOFTWARE TOOLS FOR FLEX™!

MAN
DISK/ONLY

DOCPRINT — Document/Text Formatter $55/$5
GREAT FOR YOUR MANUALS, MEMOS, LETTERS • ACCEPTS STD TEXT FILES • FORMAT EASILY CONTROLLED VIA DIRECTIONS WITHIN YOUR TEXT • OVER FORTY COMMANDS • CRT, DISK, OR PRINTER OUTPUT • OPTIONAL DRAFT REVIEW ON 64 OR 80 COL CRT • ORIGINAL MAY BE ENTIRELY UPPER CASE • MARGINS, CENTERING, INDENTATION, INSETTING, HEADER, PAGE NUMBERING, JUSTIFICATION • AUTOMATIC CAPITALIZATION • AUTOMATIC SORTED INDEX • MACHINE-LEVEL PRINTER CONTROL • BUFFERED I/O MINIMIZES DISK WEAR • EIGHT COMMAND OPTION LETTERS • SMALL FAST MACHINE LANGUAGE IMPLEMENTATION • OPTIMIZED FOR INEXPENSIVE PRINTERS • 30+ PAGE MANUAL

SCAT — Sorted Disk Directory $25/$3
EASES DISK ORGANIZATION • LISTS ALL OR SELECTED FILES IN ALPHABETICAL ORDER ON CRT OR PRINTER • DUAL 64/80 COL FORMATS • OPTIONALLY LISTS 'C' PROTECTED FILES • "WILD CARD" FILE NAMES • SHOWS VOLUME NAME/#, DISK CREATION DATE, TODAY'S DATE • DISPLAYS CREATION DATE, SIZE, DISK LOCATION, PROTECT STATUS, TYPE AND NUMBER OF FILES • DISK USAGE SUMMARY • FAST EXECUTION, 5 SEC. TYPICAL FOR 8 IN. DISK • FOUR COMMAND OPTION LETTERS

MCOPY — Multiple-File Disk Copy $30/$3
COPIES MULTIPLE FILES VIA KEYBOARD COMMAND OR "INDIRECT FILE" • REQUIRES ONLY ONE DISK DRIVE • FASTER/SIMPLER THAN STANDARD COPY • MINIMUM DISK SWAPPING • PRESERVES ORIGINAL FILES' CREATION DATES AND ATTRIBUTES

Specify 5 or 8 in. disk, FLEX-9 only. All prices postpaid. Manual price refundable on purchase of disk. Write for source code pricing. US funds only. No COD. Calif. add state tax. Foreign add 15%

CONEJO COMPUTER PRODUCTS
3655 Thousand Oaks Blvd. Suite 255
Box 5030
Westlake Village, CA 91362

FLEX is a trademark of Technical Systems Consultants, Inc.

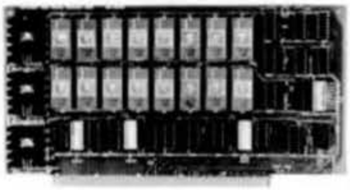
SEE PAGE 38

DIGITAL RESEARCH COMPUTERS

(214) 271-3538

32K S-100 EPROM CARD

NEW!



$79.95
KIT

USES 2716's

Blank PC Board - $34

ASSEMBLED & TESTED
ADD $30

SPECIAL: 2716 EPROM's (450 NS) Are $9.95 Ea. With Above Kit.

KIT FEATURES
1. Uses +5V only 2716 (2Kx8) EPROM's
2. Allows up to 32K of software on line!
3. IEEE S-100 Compatible
4. Addressable as two independent 16K blocks
5. Cromemco extended or Northstar bank select
6. On board wait state circuitry if needed
7. Any or all EPROM locations can be disabled
8. Double sided PC board, solder-masked, silk-screened
9. Gold plated contact fingers
10. Unselected EPROM's automatically powered down for low power
11. Fully buffered and bypassed
12. Easy and quick to assemble

32K SS-50 RAM

$329.00 KIT

For 2MHZ
Add $10

Blank PC Board
$50

For SWTPC
6800 - 6809 Buss

Support IC's
and Caps
$19.95

Complete Socket Set
$21.00

Fully Assemble,
Tested, Burned In
Add $30

NEW!

At Last! An affordable 32K Static RAM with full 6809 Capability.

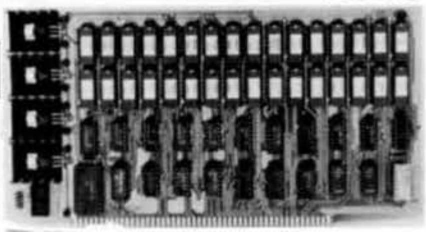
FEATURES:
1. Uses proven low power 2114 Static RAMS.
2. Supports SS50C - EXTENDED ADDRESSING
3. All parts and sockets included.
4. Dip Switch address select as a 32K block.
5. Extended addressing can be disabled.
6. Works with all existing 6800 SS50 systems.
7. Fully bypassed. PC Board is double sided, plated thru, with silk screen.

16K STATIC RAM KIT-S 100 BUSS

PRICE CUT!

$169.95 KIT

FOR 4MHZ
ADD $10



KIT FEATURES
1. Addressable as four separate 4K Blocks
2. ON BOARD BANK SELECT circuitry. (Cromemco Standard!). Allows up to 512K on line!
3. Uses 2114 (450NS) 4K Static Rams.
4. ON BOARD SELECTABLE WAIT STATES
5. Double sided PC Board, with solder mask and silk screened layout. Gold plated contact fingers.
6. All address and data lines fully buffered
7. Kit includes ALL parts and sockets.
8. PHANTOM is jumpered to PIN 67
9. LOW POWER under 1.5 amps TYPICAL from the +8 Volt Buss.
10. Blank PC Board can be populated as any multiple of 4K

BLANK PC BOARD W/DATA-$33
LOW PROFILE SOCKET SET-$12
SUPPORT IC'S & CAPS-$19.95
ASSEMBLED & TESTED-ADD $35

OUR #1 SELLING
RAM BOARD!

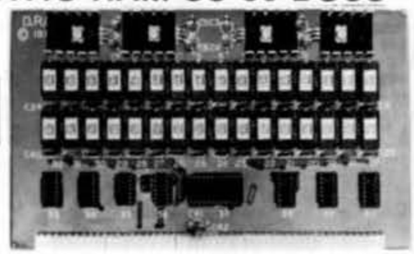
16K STATIC RAM SS-50 BUSS

PRICE CUT!

$159 KIT

FULLY STATIC!

FOR 2MHZ
ADD $10



FOR SWTPC
6800 BUSS!

ASSEMBLED AND
TESTED - $35

KIT FEATURES
1. Addressable on 16K Boundaries
2. Uses 2114 Static Ram
3. Fully Bypassed
4. Double sided PC Board. Solder mask and silk screened layout
5. All Parts and Sockets included
6. Low Power: Under 1.5 Amps Typical

BLANK PC BOARD—$35 COMPLETE SOCKET SET—$12
SUPPORT IC'S AND CAPS—$19.95

NEW! **STEREO!** NEW!

S-100 SOUND COMPUTER BOARD

At last, an S-100 Board that unleashes the full power of two unbelievable General Instruments AY3-8910 NMOS computer sound IC's. Allows you under total computer control to generate an infinite number of special sound effects for games or any other program. Sounds can be called in BASIC, ASSEMBLY LANGUAGE, etc.

COMPLETE KIT!
$84.95
(WITH DATA MANUAL)

BLANK PC
BOARD W/DATA
$31

KIT FEATURES:
- TWO GI SOUND COMPUTER IC'S
- FOUR PARALLEL I/O PORTS ON BOARD
- USES ON BOARD AUDIO AMPS OR YOUR STEREO
- ON BOARD PROTO TYPING AREA
- ALL SOCKETS, PARTS AND HARDWARE ARE INCLUDED
- PC BOARD IS SOLDERMASKED, SILK SCREENED, WITH GOLD CONTACTS
- EASY, QUICK, AND FUN TO BUILD, WITH FULL INSTRUCTIONS
- USES PROGRAMMED I/O FOR MAXIMUM SYSTEM FLEXIBILITY

Both Basic and Assembly Language Programming examples are included.

SOFTWARE:

SCL™ is now available! Our Sound Command Language makes writing Sound Effects programs a SNAP! SCL™ also includes routines for Register-Examine-Modify, Memory-Examine-Modify, and Play-Memory. SCL™ is available on CP/M* compatible diskette or 2708 or 2716. Diskette - $24.95 2708 - $19.95 2716 - $29.95. Diskette includes the source. EPROMS are ORG at E000H.

4K DYNAMIC RAM BLOWOUT!

(With Pin Out Data)

SAME AS INTEL 2107B!
4K RAMS AT AN UNBELIEVABLE 50¢ EACH!!!

Prime, new, National Semi., 1979 date coded, full spec. parts. N.S. #MM5280-5N. Same as INTEL 2107B-4, T.I. TMS4060, NEC uPD411, etc. We bought a HUGE QTY. from a West Coast Distributor at truly DISTRESS PRICES! One of the most popular and reliable RAM's ever made. These parts have been used by almost all Major Computer Main Frame Mfg. the world over! Arranged as 4K x 1, 270 NS Access Time, 22 Pin Dip. These units DO NOT use multiplexed addressing, thus making REFRESH and other timing very simple. See INTEL MEMORY DESIGN HANDBOOK for full application notes. The NAT. SEMI. MEMORY DATA BOOK is available at most Radio Shack Stores. Prime units in original factory tubes! #5280-5N 4096 BITS x 1 270 NS ACCESS

8 FOR $4.95 32 FOR $16

CRT CONTROLLER CHIP

SMC #CRT 5037. PROGRAMMABLE FOR 80 x 24, ETC. VERY RARE SURPLUS FIND. WITH PIN OUT. $12.95 EACH.

4K STATIC RAM

National Semi. MM5257. Arranged 4K x 1, +5V, 18 PIN DIP. A Lower Power, Plug in Replacement for TMS 4044, 450 NS. Several Boards on the Market Will Accept These Rams. SUPER SURPLUS PURCHASE! PRIME NEW UNITS!

8 FOR $16 32 FOR $59.95

NEW! G.I. COMPUTER SOUND CHIP

AY3-8910. As featured in July, 1979 BYTE! A fantastically powerful Sound & Music Generator. Perfect for use with any 8 Bit Microprocessor. Contains 3 Tone Channels, Noise Generator, 3 Channels of Amplitude Control, 16 bit Envelope Period Control, 2-8 Bit Parallel I/O, 3 D to A Converters, plus much more! All in one 40 Pin DIP. Super easy interface to the S-100 or other busses. $11.95 PRICE CUT!

SPECIAL OFFER: ~~$14.95~~ each Add $3 for 60 page Data Manual.

Digital Research Computers

(OF TEXAS)

P.O. BOX 401565 • GARLAND, TEXAS 75040 • (214) 271-3538

TERMS: Add $2.00 postage. We pay balance. Orders under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for insurance.

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRANTY IS AVAILABLE FREE, ON REQUEST.

*TRADEMARK OF DIGITAL RESEARCH.

WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH OF CALIFORNIA, THE SUPPLIERS OF CPM SOFTWARE.

SALE — SALE — SALE

ON HAND NOW, 80 PCS.
SPECIAL BUY!

EPSON MX-70 PRINTER

:0 char/sec, adjustable 4 to 10 inch pin feed, ; x 7 dot matrix, 96 ASCII char. set, parallel nterface, one line buffer, 10CPI or 5CPI software select), takes 3 part paper, size = I.3 x 14 x 7 inches, 12 lbs., full factory warranty.

POST PAID $375.00
NO COD's

HHH STANDARD SYNTAX GRAPHICS

Same syntax for all supported devices! Your graphics programs will run on other peoples systems, regardless of the device. Assuming standard syntax driver for same). Source, Doc and object on disk. 0S9 Modules for:

Hazelwood VC-256

255 x 250 $25.00

Watanabe 'Digi-Plot'

2000 x 3000 $25.00

These are on hand, specify 5 or 8 inch disk. Modules for Gimix Video board, Hi-Plot and others are in the works. Special while they ast, 1 VC-256 board with 0S9 diskette — 325.

We sell and support:

GIMIX (See GIMIX ad pages 3 & 56)
SMOKE SIGNAL BROADCASTING
SOUTHWEST TECHNICAL

We also know how to mix and match brands or your best performance to dollar ratio
WE PUT IT ALL TOGETHER!

H H H ENTERPRISES

P.O. BOX 493
LARUEL, MD. 20810
301-953-1155

MasterCharge VISA

0S9 is a trademark of Microware Systems Corp. and Motorola Inc.

'68' MICRO JOURNAL

★ The only ALL 6800 Computer Magazine.
★ More 6800 material than all the others combined:

MAGAZINE COMPARISON
(2 years)
Monthly Averages
6800 Articles

KB	BYTE	CC	DOBB'S	TOTAL PAGES
7.8	6.4	2.7	2.2	19.1 ea. mo.

Average cost for all four each month: $5.88
(Based on advertised 1-year subscription price)

'68' cost per month: $1.21

That's Right! Much, Much More
for About
1/5 the Cost!

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Master Charge ☐ — VISA ☐

Card # ______________ Exp. Date ______________

For ☐ 1-Year ☐ 2 Years ☐ 3 Years

Enclosed: $ ______________

Name ______________________________

Street ______________________________

City ______________ State ______________ Zip ______________

My Computer Is: ______________________________

68 MICRO JOURNAL
3018 Hamill Road
HIXSON, TN 37343

SUBSCRIPTION PRICE USA
1 year $18.50 2 years $32.50 3 years $48.50
Life subscription $250.00
NOTE: CANADA & MEXICO ADD $5.50 per year surface.
New subscriptions require 6-8 weeks processing time.

SUBSCRIPTION PRICE NON-USA (Foreign)
Sent VIA Surface Mail
1 year $30.50 2 years $56.50 3 years $84.50
Cash (USA) or drawn on a USA Bank!!!

Foreign sent VIA AIR MAIL (NON-USA)
1 year $53.50 2 years $102.50 3 years $153.50
Cash (USA) or drawn on a USA Bank!!!

If you need M68000, Z8000, or 8086 Software, take it off our shelf.

Why wait up to 6 months for high quality 16-bit resident system software, when you can get it now?

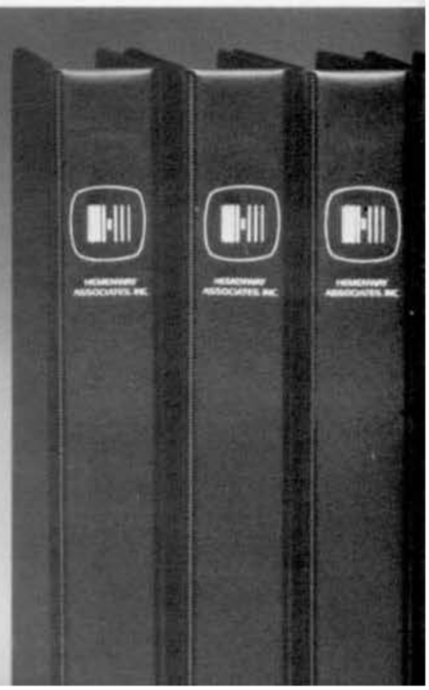
Hemenway Associates can give you the software tools you need to get the most out of your 16-bit hardware. Because 8086, Z8000 and 68000 Systems Software is sitting on our shelves right now.

Single user operating systems, macro assemblers, text editors, floating point packages, multi-tasking operating systems, linking loaders and PASCAL compilers are all ready for immediate delivery.

You've heard about Hemenway Associates' system designs in the pages of EDN. But now you can have Hemenway-designed software for your 16-bit microprocessor. Our software is easily adaptable to any hardware configuration. From manufacturers evaluation boards to full-blown microcomputers. It's efficient, easy-to-use and backed by ongoing support.

So whether you're a microcomputer manufacturer, software developer or individual microcomputer owner, Hemenway has the software for you.

For the hard facts on Hemenway software, write or call: Hemenway Associates, 101 Tremont Street, Boston, Massachusetts 02108. Phone: 617-426-1931. TWX 710321 1203. TELEX 921735.



HEMENWAY ASSOCIATES, INC.

When it comes to software, come to Hemenway

64K BYTE CMOS STATIC RAM BOARD . . . with Battery Back-Up

Using the latest in memory technology, the GIMIX 64K BYTE CMOS STATIC RAM BOARD combines the best features of previous memory boards on one board.

FULLY STATIC MEMORY with its inherent low soft error rate and freedom from alpha-particle induced errors. No complicated refresh timing or clocks required for data retention. Fully compatible with any of the 6800/6809 DMA techniques.

HIGH SPEED 200ns. memorys for guaranteed operation at 2MHz. with no wait states or clock stretching required.

ULTRA-LOW POWER CMOS RAM requires less than 1/4 AMP (250 Ma.) at 8V. for a fully populated 64K BYTE board. Less power supply loading and heat generation for cool, efficient operation.

NON-VOLATILE using an on-board nickel-cadmium battery. The board retains data even with system power removed. With the battery fully charged, the contents of the memory remain intact for a minimum of 21 days.

HIGH DENSITY permits greater memory expansion to meet the needs of todays sophisticated, multi-user/multi-tasking operating systems.

ADDRESSABLE in two 32K sections that have their own decoding for both the regular and extended (SS-50C) address lines. Each section can be addressed to any 32K boundary in the address range (1M BYTE with extended addressing). The 32K sections are divided into four 8K blocks that can be individually enabled or disabled. Disabled sections do not occupy any address space.

RELIABLE like all GIMIX products, the 64K BYTE CMOS STATIC RAM is designed with reliability in mind. Series damping resistors, a fully gridded power and ground layout, and generous power supply decoupling, all contribute to reliability and data integrity. An unsafe voltage detect circuit inhibits writes to the board, when the 8V. supply falls below a preset level, to prevent loss of data during the transition between system and battery power.

The GIMIX 64K BYTE STATIC RAM BOARD is ideally suited to a wide variety of applications.

Its high density and ultra-low power consumption make it possible to greatly expand systems with a few available bus slots and limited power supply capabilities.

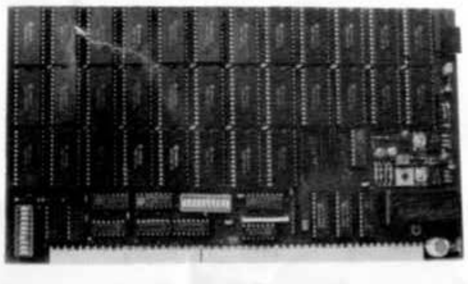
The battery back-up feature is useful where data loss due to power failure cannot be tolerated, or as a replacement for disk or tape storage where conditions such as environment prohibit their use. Since the entire board can be hardware write protected by a switch located at the top of the board, it can also be used to emulate PROM or ROM memory. This is especially useful during firmware developement where frequent software changes must be made.

When the board is used in conjunction with a device such as the GIMIX MISSING CYCLE DETECTOR BOARD, which monitors the A.C. line and generates an interrupt when a power failure occurs, critical data can be stored and system integrity maintained during either expected or unexpected power outages.

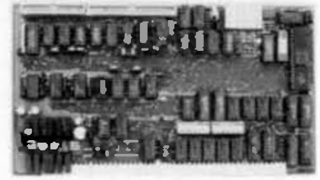
The **GIMIX 64K BYTE STATIC MEMORY BOARD** is available in 56K and 64K versions. Both version include all of the above features; gold bus connectors; and come fully assembled, burned in, and tested.

56K version ... **$ 994.56**
(Socksted for 64K)

64K version ... **$1088.64**



GIMIX KNOCKS OUT DISK PROBLEMS



GIMIX DMA DOUBLE DENSITY DISK CONTROLLER #68

The GIMIX DMA (Direct Memory Access) DISK CONTROLLER has the capabilities needed to realize the full potential of todays [illegible] multi-user/multi-tasking operating systems such as OS-9™ and UniFLEX™.

HIGH SPEED using bi-polar logic DMA circuitry for guaranteed operation at 2MHz. DMA transfers take place at full bus speed using 6809 cycle steal DMA. Once the required parameters are passed to the controller and DMA transfer is initiated the processor is free for other tasks. Interrupts can be generated to indicate the completion of the transfer.

SINGLE AND DOUBLE DENSITY data storage on any combination of 5¼" and 8" floppy disk drives, single and double headed, single and double track density, up to 4 drives total.

LOW ERROR RATES are insured by a phase lock data recovery circuit (data separator) and adjustable write precompensation circuitry for drives that require precomp. Separate precomp adjustments are provided for 5¼" and 8" drives.

ADDRESSABLE to any 8 byte boundary in the address space (1M byte when extended address decoding is used). The board occupies only 8 bytes of address space.

EXTENDED ADDRESSING control using the SS-50C extended address lines. Control of the extended address lines allows the board to perform DMA transfers to and from any address in the 1M byte address space.

FULLY BUFFERED with separate 5¼" and 8" output buffers and schmidt trigger input buffers for the disk drive signals.

The DMA controller leaves the processor free to perform other tasks once the transfer is initiated, unlike programmed I/O disk controllers which require full time use of the processor during data transfers to and from disk.

This is extremely important in a multi-user/multi-tasking environment as the processor can perform other tasks such as console I/O while a disk transfer is in progress.

#68 fully assembled, burned in, and tested ... **$548.68**

GIMIX DOUBLE DENSITY PIO DISK CONTROLLER #28

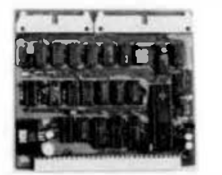
The GIMIX DOUBLE DENSITY PIO (PROGRAMMED I/O) DISK CONTROLLER is a versatile floppy disk interface for use in 6809 systems on the SS-50 or SS-50C bus. The board physically occupies one slot of the 30 pin I/O bus.

- Double the unformatted storage capacity of single density controllers
- Single and double density operation
- Phase lock data recovery circuit (data separator)
- Adjustable write precompensation (precomp)
- Controls up to four 5¼" drives
- Controls single and double headed drives
- Designed to meet the data hold-time requirements of the Western Digital 1797 floppy disk controller I.C.

The GIMIX DOUBLE DENSITY PIO DISK CONTROLLER is ideal for systems that require greater data storage than that provided by single density controllers, without increasing the number or type of drives. In most cases existing 6809 systems can be upgraded by adding only the controller and the appropriate operating system software.

#28 fully assembled, burned in and tested ... **$348.28**

GIMIX 5/8 DISK CONTROLLER BOARD #58



The GIMIX 5/8 DISK CONTROLLER is a versatile floppy disk interface for use with both 6800 and 6809 systems on the SS-50 or SS-50C bus. The board physically occupies one slot of the 30 pin I/O bus.

- Hardware and software compatible with existing disk controllers (SWTPc DC-1, DC-2 and DC-3)
- Controls up to four 5¼" drives in 6800 systems
- Controls any mix of 5¼" and 8" drives, up to four drives total, in 6809 systems
- Provides for double headed drives
- Synchronous data separator for data reliability
- Designed to meet the data hold-time requirements of the 1771 floppy disk controller I.C.

The GIMIX 5/8 DISK CONTROLLER is ideal for a variety of applications including the replacement of controllers in existing systems. [illegible] 8" drives. [illegible] operating system software.

#58 fully assembled, burned in, and tested ... **$226.58**

NOTE: When ordering disk controllers please specify the make and model of the drives being used.

GIMIX 6809 FLEX™

GIMIX versions of Technical Systems Consultants 6809 FLEX™ operating system are available for all three GIMIX disk controllers. They fully support all the features of each controller and are software compatible with other versions of FLEX™. GIMIX FLEX™ includes a disk FORMAT program that allows the user to pick the number of tracks to format, single or double sided disks, and where appropriate single or double density. GIMIX FLEX™ supports single and double track density (48 and 96 TPI) 5¼" drives and allows 96 TPI (80 track) drives to read, write, or format 48 TPI (35 or 40 track) disks.

GIMIX 6809 FLEX™ specify controller and type of drive.
8", or 5¼" 40 track (48 TPI), or 5¼" 80 track (96 TPI) ... **$90.00**

Versions of MICROWARE'S OS-9™ will be available for all three GIMIX disk controllers. Technical Systems Consultants' UniFLEX™ will be available for the GIMIX DMA controller.

GIMIX inc.

1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

FINALLY . . . .

We've talked about it.
Now, here it is.

Low power - only 30 watts
Quiet - no fans, brushless motor
Incredibly small - like minifloppy
Fast - step rate 1.5 mSec or better
- rotation 3600 rpm

"FLEX" is a trademark of Technical Systems Consultants

5 Megabytes, formatted
Software for FLEX, UCSD Pascal available separately
Complete subsystem, includes:
- ST-506 Winchester drive
- controller/formatter
- SS-50 interface
- anodized cabinet with power supply
- cables

Tallgrass Technologies Corporation

7623 W. 86th St.
P.O. Box 12047
Overland Park, Kansas 66212
(913) 381-5588

Price $2295. Terms C.O.D. or Prepaid
Introductory Price—Direct Mail Sales Only
Shipping charges will be added.

Supply of drives limited. Delivery estimated at 3 to 6 weeks ARO, first come first served.

Control Systems, Inc.'s license to distribute the UCSD p-system has been acquired by Tallgrass Technologies Corp., effective January 31, 1981. Tallgrass Technologies Corp. is the only authorized distributor of UCSD Pascal for the 6809 SS-50 bus. Tallgrass Technologies Corp. will endeavor to provide assistance to existing CSI customers.

UCSD PASCAL

FREE MEMBERSHIP IN USUS USER GROUP
FREE* USER GROUP SOFTWARE**

*Free when written request accompanies pre-paid order. Some files may be omitted at the discretion of the UCSD System User Society and/or Tallgrass Technologies Corp.
**Supplied as source files only, in as-is condition, as received from USUS Library.

COMBINE	KB DATABASE
DISKREAD	KB DEMO
FORMAT	KB SCUNIT
L	KB STARTER
TYPESET	KB TESTDB
PRETTY	COMPARE
H14 DRIVER	COMPRESS
H19 DOC	INDEX
H19 GOTOXY	WUMPUS & TEACH WUMPUS & CAVES
H19 MISCINFO	GETSORT
HAZEL MISCINFO	STRUCT
HEXOUT	UPDATE
LINECOUNTER	ADDRS DOC
PE1100 GOTOXY	CRTINPUT
PERUSE	DIR
PTP & PTP DOC	GETNUMBER
SHELLMSORT	HEXOECOCT
SMARTREMOT	1D2ID
TVI912C GOTOXY	MAKEMASKS
WRITER & WRITER DOC	PEEK POKE
BLACKJACK	QUICKSORT
CHASE	SCREENCNTL
OTHELLO	SOFT TOOLS DOC
DEBTS & STORE DATA	SP TEXT
SNOOPY	UNIT GOOD
DBBUILDER TEXT	
DBUNIT 1, 2, 3, 4	

"UCSD P-system" and "UCSD Pascal" are registered trademarks of the Regents of the University of California.

SOFTWARE FURNISHED:
UCSD PASCAL
VERII.0

OP/SYS
PASCAL COMPILER
SYSTEM FILER
SYSTEM LINKER
SYSTEM LIBRARY
LIBRARIAN
YALOE
SCREEN EDITOR
L2 EDITOR
6809 MACRO ASSEM
6800 MACRO ASSEM
6502 MACRO ASSEM
Z-80 MACRO ASSEM
P-CODE DISSASSEMBLER
MARK DUPDIR
COPY DUPDIR
PATCH
SETUP
BINDER
FLIP
NEW DISK 512 formatting program

- High-performance 512-byte/sector disk formats.
- All UCSD software is packaged on a single 8" diskette, no more CSI-1, -2, -3.
- User-group software available free on written request with order.

HARDWARE SUPPORTED:*
GIMIX 6809 CPU
GIMIX 5/8 CONTROLLER
SWTPC 6809 CPU
SSB 6809 CPU
SWTPC DC3 5¼"
SWTPC DC2 5¼"
SSB DCB-4 8"
SSB DCB-4a
SWTPC DMAF-1 8"
SWTPC DMAF-2 8"
SWTPC 14" WINCHESTER
TALLGRASS 5¼" WINCHESTER
SSB BFD-68 8"
*Requires 56k of memory; 6809 CPU

HOW TO ORDER:
PRICE $459 US

Prepaid orders only.

Written orders preferred.

VISA & Mastercharge acceptable.

Overseas orders add $20 US for extra postage and handling.

Tallgrass Technologies Corporation

7623 W. 86th St.
P.O. Box 12047
Overland Park, Kansas 66212
(913) 381-5588

'68' Micro Journal
3018 Hamill Rd.
Hixson, TN 37343

Second Class Postage Paid
At Chattanooga, TN
ISSN 0194-5025

TRENTON
BOX 708
MR. MICKEY FERGUSON
000422 A/E
GA 30752
MJ

3

SOFTWARE FOR THE HARDCORE

tFORTH™ THE PROFESSIONAL'S CHOICE

WHAT? FORTH! WHO? TALBOT!

Extended version of the original 6809 implementation of FORTH INTEREST GROUP FORTH by astrophysicist Dr. Ray Talbot; equivalent 6800 version also available. Now obtain tFORTH direct from the author, thereby ensuring direct and quick support. Supplied on FLEX™ soft-sectored diskettes, although when you discover the elegant simplicity of FORTH you will use it as a self-contained system!

WHO USES IT? CRAFTSMEN!

A master furniture craftsman like Chippendale used special tools to ply his craft; much better tools than you would supply to a high school industrial arts class. It has been said that if Chippendale had made programs rather than furniture he would have used FORTH as his tool. He would have not tolerated Beginner's All-purpose Symbolic Widget Instructional Toys.

If you want to learn how to program, use a language designed for teaching — PASCAL or BASIC. If you know how to program, use a language designed for craftsman — FORTH.

FORTH applications span a wide range of tasks. It is ideal for laboratory instrument control, data acquisition and analysis, process control, interactive systems, and real-time systems. It has been used for Astronomy through Zoology with the practical worlds of aircraft simulations, automated banking, and computerized bulletin boards in between.

WHY USE IT? TIME=MONEY!

Users of FORTH report productivity gains of 2 to 10 over their other development tools. FORTH develops faster and runs faster than most BASICs or PASCALs (see time comparisons in '68' Micro Journal, 1981, Feb. p.14, April p.14 (compare equivalent algorithms!), and May p.27.

firmFORTH™ produces equally fast, but much more compact code. It is simple to develop and test complex systems using tFORTH, then use firmFORTH to produce a compact rommable product.

tFORTH SYSTEM AND APPLICATIONS

THE PROFESSIONAL's CHOICE!

Compatible with all FLEX systems: GIMIX, SWTP, SSB, or EXORcisor; easy to convert to other operating systems or make standalone.

Specify 5 or 8 inch diskette and specify 6800 or 6809.

Manuals available seperately - price in (), deductable from system ordered later.

tFORTH $100 ($15)

Basic complete FORTH system with text Editor, debugging tools, and vocabulary to interface with FLEX.

tFORTH+ $250 ($25)

tFORTH plus 2nd screen editor, full macro assembler, extended data types, CASEs, additional diskettes with GOING FORTH (a computer aided instruction course on FORTH written in FORTH) and GOODIES #1 (a collection of debugging and documentation tools, games, execution simulator, generalized formatted numerical input, and miscellaneous utilities). Individual parts of tFORTH+ are available for previous purchasers of tFORTH.

APPLICATIONS PROGRAMS FOR USE WITH tFORTH

firmFORTH $350 ($10)

For doing target compilations to rommable code. Ideal for developing proms for instruments or even complete operating systems! Automatically deletes unused code and unneeded dictionary information. New version 3.0 can compile directly to memory or to disk storage (the latter permits development of larger programs), and contains code for developing interupt service routines. Includes full source code for target compiler as well as the essential portions of the full FORTH nucleus. Requires but does not include tFORTH+.

Tiny PASCAL in FORTH Data Base Management Tool Kit and other applications — write for details.

™ tFORTH is a TradeMark of Talbot Microsystems
™ FLEX is a TradeMark of Technical Systems Consultants

Talbot

MICROSYSTEMS

5030 Kensington Way, Riverside, California 92507, (714) 781-0464